CHƯƠNG 1: CÁC KIẾN THỨC CƠ BẢN VỀ MẠNG MÁY TÍNH

1.1. Mô hình tham khảo 7 tầng OSI

Mô hình kết nối hệ thống mở được Tổ chức quốc tế về tiêu chuẩn hoá ISO (International Organizaiton for Standardization) đưa ra nhằm cung cấp một mô hình chuẩn cho các nhà sản xuất và cung cấp sản phẩm viễn thông áp dụng theo để phát triển các sản phẩm viễn thông. Ý tưởng mô hình hoá được tạo ra còn nhằm hỗ trợ cho việc kết nối giữa các hệ thống và modun hoá các thành phần phục vụ mạng viến thông.

a. Chức năng của mô hình OSI:

- Cung cấp kiến thức về hoạt động của kết nối liên mạng
- Đưa ra trình tự công việc để thiết lập và thực hiện một giao thức cho kết nối các thiết bị trên mạng.

Mô hình OSI còn có một số thuận lợi sau :

- Chia nhỏ các hoạt động phức tạp của mạng thành các phần công việc đơn giản.
- Cho phép các nhà thiết kế có khả năng phát triển trên từng modun chức năng.
- Cung cấp các khả năng định nghĩa các chuẩn giao tiếp có tính tương thích cao "plug and play" và tích hợp nhiều nhà cung cấp sản phẩm.

b. Cấu trúc mô hình OSI:

Mô hình OSI gồm 7 lớp (level), mỗi lớp thực hiện các chức năng riêng cho hoạt động kết nối mạng.

Hình 1-1 Mô tả bẩy lớp OSI. 4 lớp đầu định nghĩa cách thức cho đầu cuối thiết lập kết nối với nhau để trao đổi dữ liệu. 3 lớp trên dùng để phát triển các ứng dụng để đầu cuối kết nối với nhau và người dùng.

Application (Upper Layer)	Aplication	
	Presentation	
	Session	
	Transport Layer	Data Lower Layer
	Network Layer	
	Data Link	
	Physical	

Các lớp trên

3 lớp trên cùng của mô hình OSI thường được gọi là các lớp ứng dụng (Application layers) hay còn gọi là các lớp cao. Các lớp này thường liên quan tới giao tiếp với người dùng, định dạng của dữ liệu và phương thức truy nhập các ứng dụng đó.

Hình 1-2 Mô tả các lớp trên và cung cấp thông tin với các chức năng của nó qua ví dụ:

	- Lớp ứng dụng: chức năng giao	Telnet, HTTP

Application	tiếp giữa người sử dụng và các chương trình ứng dụng	
Presentation	- Lớp trình bày: cách thức chuẩn hoá dữ liệu và trình bày số liệu - Có chức năng đặc biệt là mã hoá dữ liệu người sử dung	ASSCII EBCDIC JPEC
Session	- Lớp phiên: thiết lập, duy trì và huỷ bỏ một phiên làm việc	NFS, SQL
Transport Layer		
Network Layer		
Data Link		
Physical		

- Application layer : đây là lớp cao nhất trong mô hình. Nó là nơi mà người sử dụng hoặc kết nối các chương trình ứng dụng với các thủ tục cho phép truy nhập vào mạng.
- Presentation layer : Lớp presentation cung cấp các mã và chức năng để chuyển đổi mà được cung cấp bởi lớp ứng dụng. Các chức năng đó đảm bảo rằng dữ liệu từ lớp ứng dụng trong một hệ thống có thể được đọc bởi lớp ứng dụng của một hệ thống khác. VD : dùng để mã hoá dữ liệu từ lớp ứng dụng : như mã hoá ảnh jpeg , gif. Mã đó cho phép ta có thể hiện lên trang web .
- Session layer : được sử dụng để thiết lập, duy trì và kết thúc phiên làm việc giữa các lớp presentation. Việc trao đổi thông tin ở lớp này bao gồm yêu cầu dịch vụ và đáp ứng yêu cầu của các ứng dụng trên thiết bị khác.

Các lớp dưới.

4 lớp dưới của mô hình OSI sử dụng để định nghĩa làm thế nào để dữ liệu được truyền đi trong các dây nối vật lý, các thiết bị mạng và đi đến trạm đầu cuối cuối cùng là đến các lớp ứng dụng. Quấn sách này ta chỉ quan tâm đến 4 lớp cuối. Và sẽ xem xét từng lớp một cách chi tiết giao thiếp giữa các lớp trong mô hình OSI:

Sử dụng phương pháp protocal stack để kết nối giữa hai thiết bị trong mạng. Protocal stack là một tập hợp các quy định dùng để định nghĩa làm thế nào để dữ liệu truyền qua mạng.

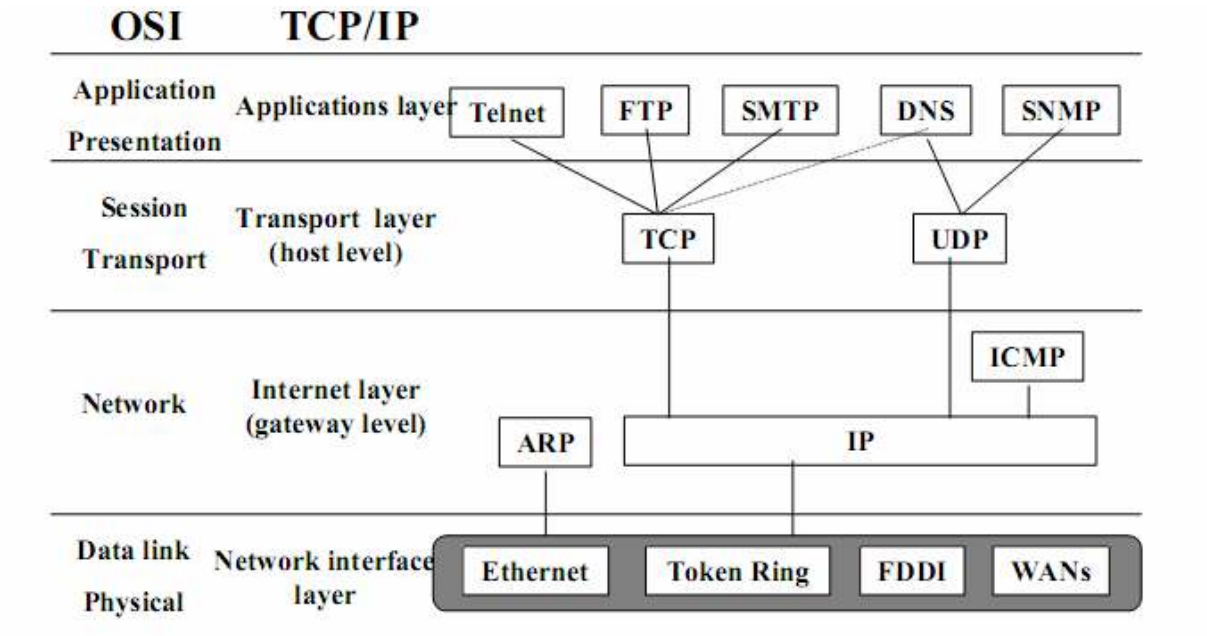
Ví dụ với : TCP/IP mỗi Layer cho phép dữ liệu truyền qua. Các lớp đó trao đổi các thông tin để cung cấp cuộc liên lạc giữa hai thiết bị trong mạng. Các lớp giao tiếp với nhau sử dụng Protocal Data Unit (PDU). Thông tin điều khiển của PDU được thêm

vào với dữ liệu ở lớp trên. Và thông tin điều khiển này nằm trong trường gọi là trường header và trailer.

Hình 1-3 Data encapsulation

			Application	
			Presentation	
	Upper Layer Data		Session	
TCP Header	Upper Layer Data		Transport	Segment
IP Header	Data		Network	Packet
LLC Header	Data	FCS	Data Link	Frame
MAC Header	Data	FCS	Physical	Bits
01011101010010000 10				

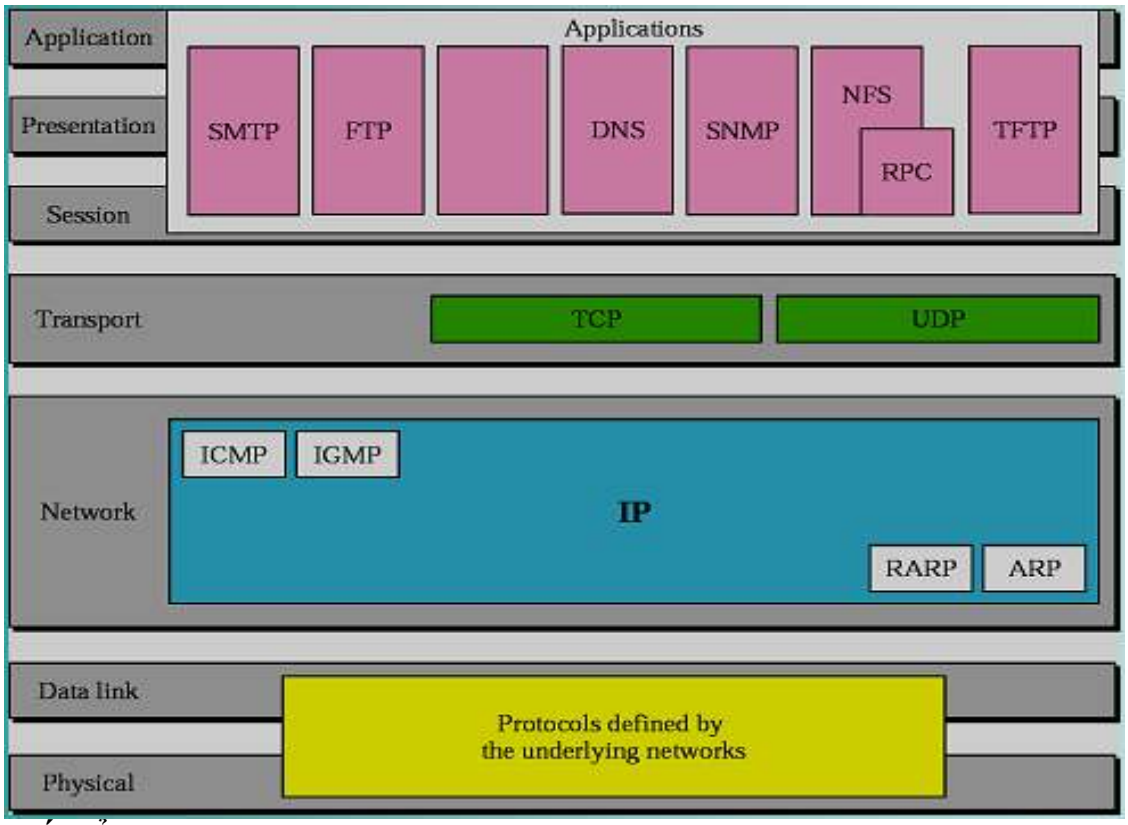
1.2. Họ giao thức TCP/IP



Các tầng của giao thức TCP/IP so với cấc tầng của mô hình OSI

Application: Xác nhận quyền, nén dữ liệu và các dịch vụ cho người dùng
Transport: Xử lý dữ liệu giữa các hệ thống va cung cấp việc truy cập mạng cho các ứng dụng
Network: Tìm đường cho các packet

Link: Mức OS hoặc các thiết bị giao tiếp mạng trên một máy tính



Một số điểm khác nhau của TCP/IP và mô hình OSI

+ Lớp ứng dụng trong TCP/IP xử lý chức năng của lớp 5,6,7 trong mô hình OSI

+ Lớp transport trong TCP/IP cung cấp cớ chế UDP truyền không tin cậy, transport trong OSI luôn đảm bảo truyền tin cậy

+ TCP/IP là một tập của các protocols (một bộ giao thức)

+ TCP/IP xây dựng trước OSI

Quy trình đóng gói dữ liệu trong mô hình TCP/IP như sau:

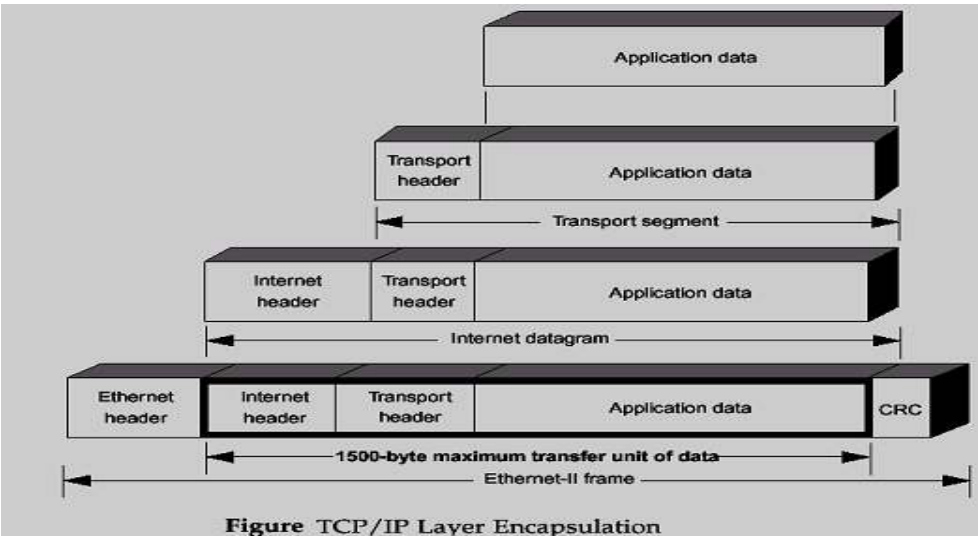
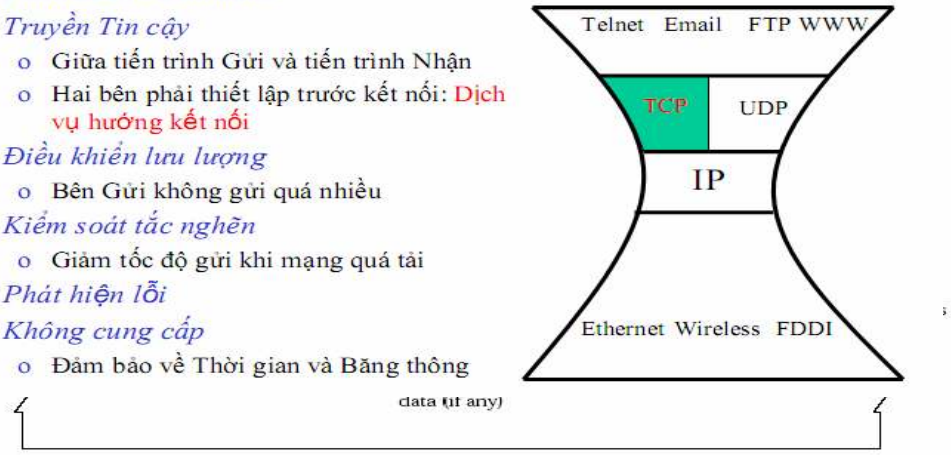


Figure TCP/IP Layer Encapsulation

1.3. So sánh giữa hai giao thức TCP và UDP

Tầng Giao vận : Dịch vụ TCP

- *Phân kênh / Dồn kênh*
- *Truyền Tin cậy*
  - Giữa tiến trình Gửi và tiến trình Nhận
  - Hai bên phải thiết lập trước kết nối: Dịch vụ hướng kết nối
- *Điều khiển lưu lượng*
  - Bên Gửi không gửi quá nhiều
- *Kiểm soát tắc nghẽn*
  - Giảm tốc độ gửi khi mạng quá tải
- *Phát hiện lỗi*
- *Không cung cấp*
  - Đảm bảo về Thời gian và Băng thông



1.4. Cổng giao thức

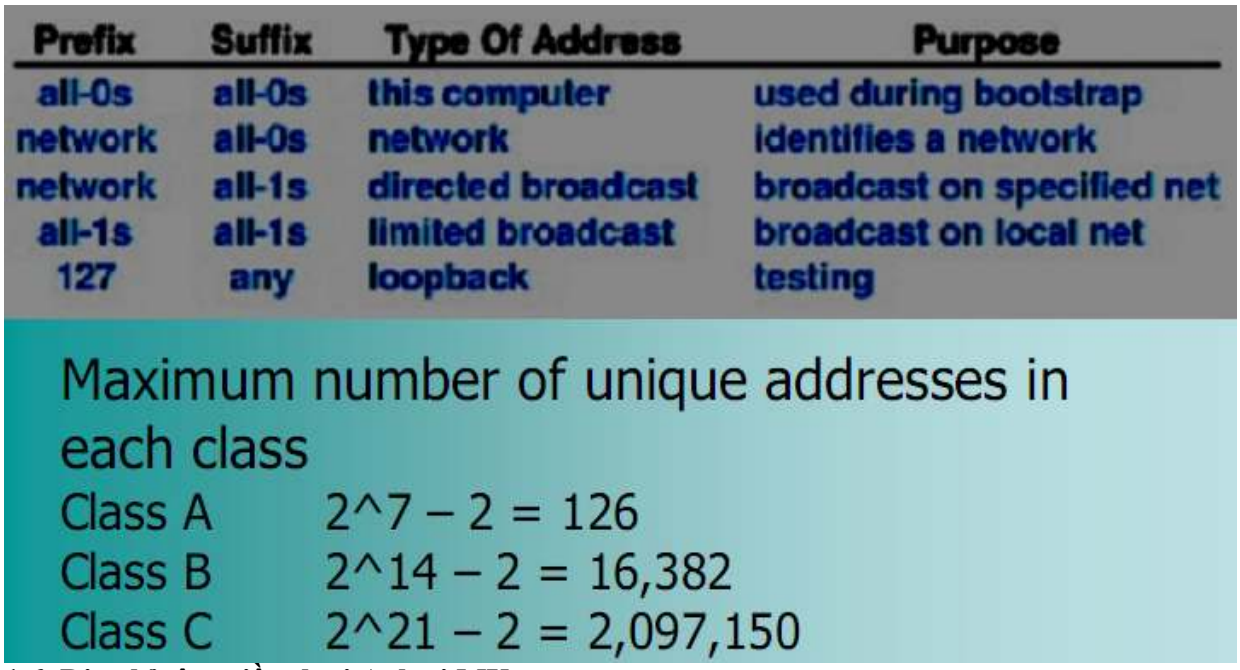
Là một số năm trong khoảng 1..65535 dùng để phân biệt giữa 2 ứng dụng mạng với nhau gắn với địa chỉ IP và Socket

Một số cổng và các giao thức thông dụng:

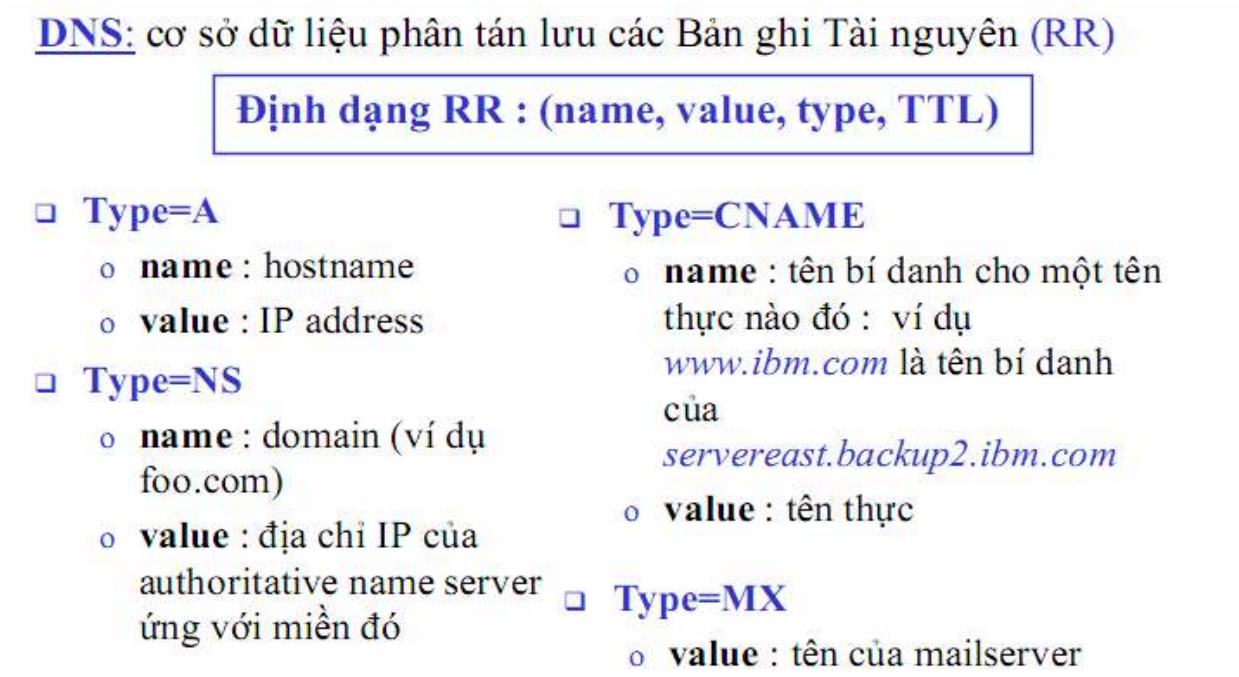
+ FTP: 21
+ Telnet: 23
+ SMTP: 25
+ POP3: 110
+ HTTP:80

1.5. Địa chỉ IP, các địa chỉ IP dành riêng

	0	1	2	3	4				8	16	24	
Class A	**0**	**Netid**							**Hostid**			
Class B	**1**	**0**	**Netid**							**Hostid**		
Class C	**1**	**1**	**0**	**Netid**							**Hostid**	
Class D	**1**	**1**	**1**	**0**	**Multicast address**							
Class E	**1**	**1**	**1**	**1**	**0**	**Reverved for future use**						



1.6. Địa chỉ tên miền: loại A, loại MX..



1.7. Một số giao thức ở tầng ứng dụng: HTTP, SMTP, POP3, FTP...

- Chúng ta sẽ nghiên cứu chi tiết các giao thức này ở chương 3

CHƯƠNG 2: LẬP TRÌNH MẠNG TRONG .NET FRAMEWORK

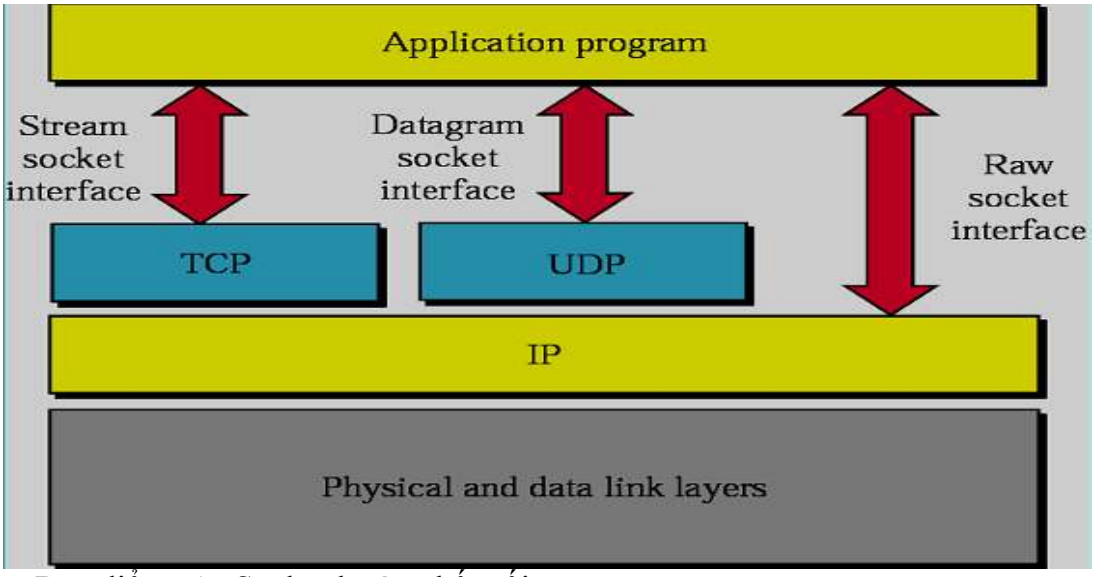
2.1. Socket hướng kết nối (TCP Socket)

Socket là một giao diện lập trình ứng dụng (API) mạng

Thông qua giao diện này chúng ta có thể lập trình điều khiển việc truyền thông giữa hai máy sử dụng các giao thức mức thấp là TCP, UDP…

Socket là sự trừu tượng hoá ở mức cao, có thể tưởng tượng nó như là thiết bị truyền thông hai chiều gửi – nhận dữ liệu giữa hai máy tính với nhau.

- Các loại Socket
  - Socket hướng kết nối (TCP Socket)
  - Socket không hướng kết nối (UDP Socket)
  - Raw Socket



- Đặc điểm của Socket hướng kết nối
  - Có 1 đường kết nối ảo giữa 2 tiến trình
  - Một trong 2 tiến trình phải đợi tiến trình kia yêu cầu kết nối.
  - Có thể sử dụng để liên lạc theo mô hình Client/Server
  - Trong mô hình Client/Server thì Server lắng nghe và chấp nhận một yêu cầu kết nối
  - Mỗi thông điệp gửi đều có xác nhận trở về
  - Các gói tin chuyển đi tuần tự
- **Đặc điểm của Socket không hướng kết nối**
  - **Hai tiến trình liên lạc với nhau không kết nối trực tiếp**
  - **Thông điệp gửi đi phải kèm theo địa chỉ của người nhận**
  - **Thông điệp có thể gửi nhiều lần**
  - **Người gửi không chắc chắn thông điệp tới tay người nhận**
  - **Thông điệp gửi sau có thể đến đích trước thông điệp gửi trước đó.**
- Số hiệu cổng của Socket

- Để có thể thực hiện các cuộc giao tiếp, một trong hai quá trình phải công bố số hiệu cổng của socket mà mình sử dụng.
- Mỗi cổng giao tiếp thể hiện một địa chỉ xác định trong hệ thống. Khi quá trình được gán một số hiệu cổng, nó có thể nhận dữ liệu gởi đến cổng này từ các quá trình khác.
- Quá trình còn lại cũng yêu cầu tạo ra một socket.

2.1.1. Giới thiệu về NameSpace System.Net và System.Net.Sockets

- Cung cấp một giao diện lập trình đơn giản cho rất nhiều các giao thức mạng.
- Có rất nhiều lớp để lập trình
- Ta quan tâm lớp IPAdress, IPEndPoint, DNS, …
- Lớp IPAdress
- Một số Field cần chú ý:
  - Any: Cung cấp một địa chỉ IP để chỉ ra rằng Server phải lắng nghe trên tất cả các Card mạng
  - Broadcast: Cung cấp một địa chỉ IP quảng bá
  - Loopback: Trả về một địa chỉ IP lặp
  - AdressFamily: Trả về họ địa chỉ của IP hiện hành
- Lớp IPAddress
- Một số phương thức cần chú ý:
  - Phương thức khởi tạo
    - IPAddress(Byte[])
    - IPAddress(Int64)
  - IsLoopback: Cho biết địa chỉ có phải địa chỉ lặp không
  - Parse: Chuyển IP dạng xâu về IP chuẩn
  - ToString: Trả địa chỉ IP về dạng xâu
  - TryParse: Kiểm tra IP ở dạng xâu có hợp lệ không?
- Lớp IPEndPoint
- Một số phương thức cần chú ý:
  - Phương thức khởi tạo
    - IPEndPoint (Int64, Int32)
    - IPEndPoint (IPAddress, Int32)
  - Create: Tạo một EndPoint từ một địa chỉ Socket
  - ToString : Trả về địa chỉ IP và số hiệu cổng theo khuôn dạng ĐịaChỉ: Cổng, ví dụ: 192.168.1.1:8080
- Lớp DNS
- Một số thành phần của lớp:
  - HostName: Cho biết tên của máy được phân giải
  - GetHostAddress: Trả về tất cả IP của một trạm
  - GetHostEntry: Giải đáp tên hoặc địa chỉ truyền vào và trả về đối tượng IPHostEntry

- GetHostName: Lấy về tên của máy tính cục bộ
- NameSpace System.Net.Sockets
  - Một số lớp hay dùng: TcpClient, UdpClient, TcpListener, Socket, NetworkStream, …
- Để tạo ra Socket
  - Socket(AddressFamily *af*, SocketType *st*, ProtocolType *pt*)

SocketType	**Protocoltype**	**Description**
Dgram	Udp	Connectionless communication
Stream	Tcp	Connection-oriented communication
Raw	Icmp	Internet Control Message Protocol
Raw	Raw	Plain IP packet communication

```
using System.Net;
using System.Net.Sockets;
   class SockProp    {
      public static void Main()       {
         IPAddress ia = IPAddress.Parse("127.0.0.1");
         IPEndPoint ie = new IPEndPoint(ia, 8000);
         Socket test = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
         Console.WriteLine("AddressFamily: {0}", test.AddressFamily);
         Console.WriteLine("SocketType: {0}", test.SocketType);
         Console.WriteLine("ProtocolType: {0}",  test.ProtocolType);
         Console.WriteLine("Blocking: {0}", test.Blocking);
         test.Blocking = false;
         Console.WriteLine("new Blocking: {0}", test.Blocking);
         Console.WriteLine("Connected: {0}", test.Connected);
         test.Bind(ie);
         IPEndPoint iep = (IPEndPoint)test.LocalEndPoint;
         Console.WriteLine("Local EndPoint: {0}", iep.ToString());
         test.Close();
         Console.ReadKey();
      }
   }
```

2.1.2. Viết chương trình cho phía máy chủ

- Viết chương trình cho phía máy chủ
  - Tạo một Socket
  - Liên kết với một IPEndPoint cục bộ
  - Lắng nghe kết nối
  - Chấp nhận kết nối
  - Gửi nhận dữ liệu theo giao thức đã thiết kế

- Đóng kết nối sau khi đã hoàn thành và trở lại trạng thái lắng nghe chờ kết nối mới

- Viết chương trình cho phía máy chủ

```
IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);
Socket newsock = Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
newsock.Bind(ipep);
newsock.Listen(10);
Socket client = newsock.Accept();
//Gửi nhận dữ liệu theo giao thức đã thiết kế
…….
newsock.Close();
```

Chương trình Server:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
class Server{
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2008);
      Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
      server.Bind(iep);
      server.Listen(10);
      Console.WriteLine("Cho ket noi tu client");
      Socket client = server.Accept();
      Console.WriteLine("Chap nhan ket noi tu:{0}",
client.RemoteEndPoint.ToString());
      string s = "Chao ban den voi Server";
      //Chuyen chuoi s thanh mang byte
      byte[] data = new byte[1024];
      data = Encoding.ASCII.GetBytes(s);
      //gui nhan du lieu theo giao thuc da thiet ke
      client.Send(data,data.Length,SocketFlags.None);
      while (true) {
         data = new byte[1024];
         int recv = client.Receive(data);
         if (recv == 0) break;
         //Chuyen mang byte Data thanh chuoi va in ra man hinh
         s = Encoding.ASCII.GetString(data, 0, recv);
         Console.WriteLine("Clien gui len:{0}", s);
         //Neu chuoi nhan duoc la Quit thi thoat
         if (s.ToUpper().Equals("QUIT")) break;
         //Gui tra lai cho client chuoi s
         s = s.ToUpper();
         data = new byte[1024];
```

```
            data = Encoding.ASCII.GetBytes(s);
            client.Send(data, data.Length, SocketFlags.None);
        }
        client.Shutdown(SocketShutdown.Both);
        client.Close();
        server.Close();
    }
}
```

2.1.3. Viết chương trình cho phía máy khách

- Viết chương trình cho phía máy khách
  - Xác định địa chỉ của Server
  - Tạo Socket
  - Kết nối đến Server
  - Gửi nhận dữ liệu theo giao thức đã thiết kế
  - Đóng Socket
- Viết chương trình cho phía máy khách

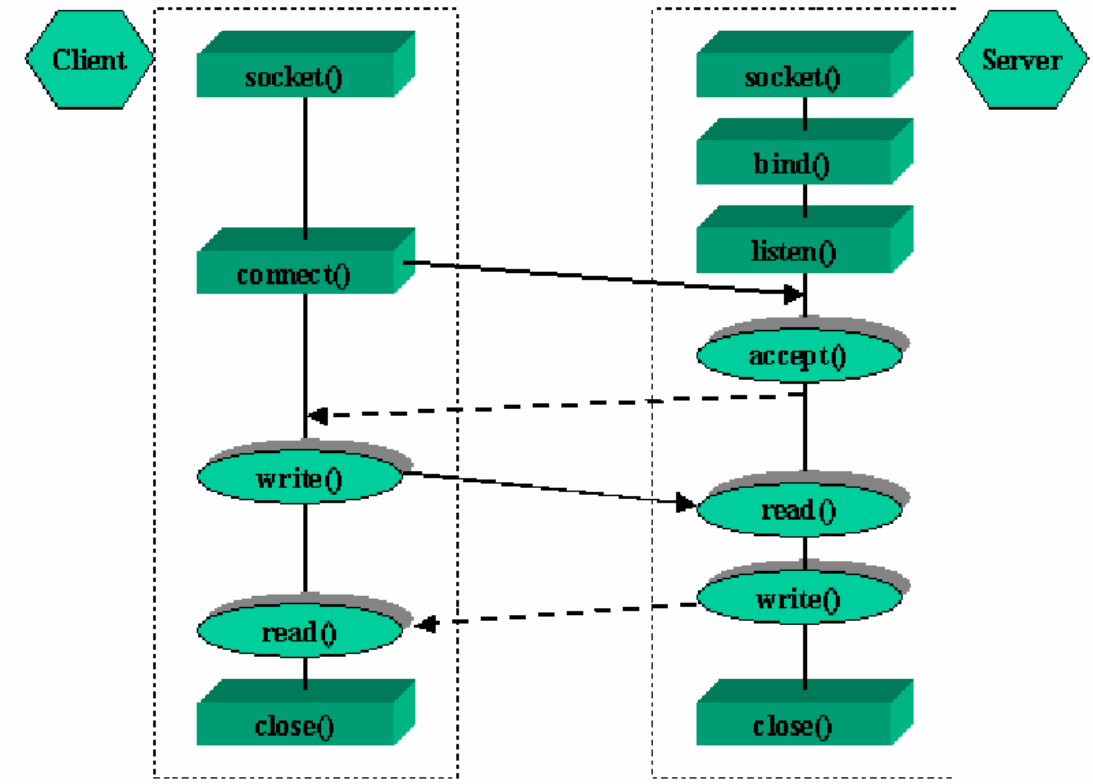
```
IPEndPoint ipep = new IPEndPoint(Ipaddress.Parse("192.168.1.6"), 9050);
Socket server = new Socket(AddressFamily.InterNetwork,SocketType.Stream,
                ProtocolType.Tcp);
server.Connect(ipep);
```

Chương trình Client:

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

class Client {
    static void Main(string[] args) {
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2008);
        Socket client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        client.Connect(iep);
        byte[] data = new byte[1024];
        int recv = client.Receive(data);
        string s = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine("Server gui:{0}", s);
        string input;
        while (true) {
            input = Console.ReadLine();
            //Chuyen input thanh mang byte gui len cho server
            data = new byte[1024];
            data = Encoding.ASCII.GetBytes(input);
            client.Send(data, data.Length, SocketFlags.None);
            if (input.ToUpper().Equals("QUIT")) break;
            data = new byte[1024];
            recv = client.Receive(data);
```

```
            s = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine("Server gui:{0}", s);
        }
        client.Disconnect(true);
        client.Close();
    }
}
```



2.1.4. Sử dụng các luồng nhập xuất với Socket

Từ Socket ta có thể tạo ra luồng để nhập xuất với Socket đó là sử dụng lớp NetworkStream

Property	**Description**
CanRead	Is true if the NetworkStream supports reading
CanSeek	Is always false for NetworkStreams
CanWrite	Is true if the NetworkStream supports writing
DataAvailable	Is true if there is data available to be read

Ví dụ chương trình Client/Server sử dụng NetworkStream để gửi và nhận dữ liệu
Chương trình Client sử dụng NetworkStream:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
```

```
class Program {
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2009);
      Socket client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
      client.Connect(iep);
      NetworkStream ns = new NetworkStream(client);
      byte[] data = new byte[1024];
      while (true) {
         string input = Console.ReadLine();
         data = Encoding.ASCII.GetBytes(input);
         ns.Write(data, 0, data.Length);
         if (input.ToUpper().Equals("QUIT")) break;
         data = new byte[1024];
         int rec = ns.Read(data, 0, data.Length);
         string s = Encoding.ASCII.GetString(data, 0, rec);
         Console.WriteLine(s);
      }
      client.Close();
   }
}
```

Chương trình Server sử dụng NetworkStream:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
class Program {
   static void Main(string[] args) {
      IPEndPoint iep=new IPEndPoint(IPAddress.Parse("127.0.0.1"),2009);
      Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
      server.Bind(iep);
      server.Listen(10);
      Socket client = server.Accept();
      byte[] data;
      NetworkStream ns = new NetworkStream(client);
      while (true) {
         data = new byte[1024];
         int rec = ns.Read(data, 0, data.Length);
         string s = Encoding.ASCII.GetString(data, 0, rec);
         Console.WriteLine(s);
         data = new byte[1024];
         s = s.ToUpper();
         if (s.Equals("QUIT")) break;
         data = Encoding.ASCII.GetBytes(s);
         ns.Write(data, 0, data.Length);
      }
```

```
        client.Close();
        server.Close();
    }
}
```

Trên cở sở của NetworkStream ta có thể nối thêm các luồng để nhập xuất theo hướng ký tự như StreamReader, StreamWriter

Sau đây là một ví dụ về chương trình Client/Server sử dụng luồng nhập xuất, chương trình Server chép phép Client gửi lên yêu cầu, nếu yêu cầu là GetDate không phân biệt chữ hoa chữ thường thì Server trả về cho Client ngày hiện tại, nếu yêu cầu là GetTime không phan biệt hoa thường thì Server trả về giờ hiện tại, nếu là Quit thì Server ngắt kết nối với Client, không phải các trường hợp trên thì thông báo không hiểu lênh.

Chương trình phía Client:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;
class DateTimeClient {
    static void Main(string[] args) {
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9999);
        Socket client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                                                                ProtocolType.Tcp);
        client.Connect(iep);
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        while (true) {
            string input = Console.ReadLine();
            sw.WriteLine(input);
            sw.Flush();
            if (input.ToUpper().Equals("QUIT")) break;
            string kq = sr.ReadLine();
            Console.WriteLine(kq);
        }
        sr.Close();
        sw.Close();
        ns.Close();
        client.Close();
    }
}
```

Chương trình phía Server:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
```

```
using System.Net.Sockets;
using System.IO;
class DateTimeServer{
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2009);
      Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                                                              ProtocolType.Tcp);
      server.Bind(iep);
      server.Listen(10);
      Socket client = server.Accept();
      NetworkStream ns = new NetworkStream(client);
      StreamReader sr = new StreamReader(ns
      StreamWriter sw = new StreamWriter(ns);
      string kq="";
      while (true) {
         string s = sr.ReadLine();
         s=s.ToUpper();
         if (s.Equals("QUIT")) break;
         if (s.Equals("GETDATE"))
            kq = DateTime.Now.ToString("dd/MM/yyyy");
         else
            if (s.Equals("GETTIME"))
               kq = DateTime.Now.ToString("hh:mm:ss");
            else
               kq = "Khong hieu lenh";
         sw.WriteLine(kq);
         sw.Flush();
      }
      sr.Close();
      sw.Close();
      client.Close();
   }
}
```

2.2. Socket không hướng kết nối (UDP Socket)

- Chương trình phía máy chủ
  - Tạo một Socket
  - Liên kết với một IPEndPoint cục bộ
  - Gửi nhận dữ liệu theo giao thức đã thiết kế
  - Đóng Socket
- Chương trình phía máy khách
  - Xác định địa chỉ Server
  - Tạo Socket
  - Gửi nhận dữ liệu theo giao thức đã thiết kế
  - Đóng Socket

2.2.1. Viết chương trình cho phía máy chủ

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
class Program {
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2008);
      Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
ProtocolType.Udp);
      server.Bind(iep);
      //tao ra mot Endpot tu xa de nhan du lieu ve
      IPEndPoint RemoteEp = new IPEndPoint(IPAddress.Any, 0);
      EndPoint remote=(EndPoint)RemoteEp;
      byte[] data = new byte[1024];
      int recv = server.ReceiveFrom(data, ref remote);
      string s = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine("nhan ve tu Client:{0}", s);
      data = Encoding.ASCII.GetBytes("Chao client");
      server.SendTo(data, remote);
      while (true) {
         data=new byte[1024];
         recv = server.ReceiveFrom(data, ref remote);
         s = Encoding.ASCII.GetString(data, 0, recv);
         if (s.ToUpper().Equals("QUIT")) break;
         Console.WriteLine(s);
         data=new byte[1024];
         data=Encoding.ASCII.GetBytes(s);
         server.SendTo(data,0,data.Length,SocketFlags.None,remote);
      }
      server.Close();
   }
}
```

2.2.2. Viết chương trình cho phía máy khách

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

class Program {
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2008);
      Socket client = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
ProtocolType.Udp);
      String s = "Chao server";
      byte[] data = new byte[1024];
      data = Encoding.ASCII.GetBytes(s);
      client.SendTo(data, iep);
      EndPoint remote = (EndPoint)iep;
```

```
        data = new byte[1024];
        int recv = client.ReceiveFrom(data, ref remote);
        s = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine("Nhan ve tu Server{0}",s);
        while (true) {
            s = Console.ReadLine();
            data=new byte[1024];
            data = Encoding.ASCII.GetBytes(s);
            client.SendTo(data, remote);
            if (s.ToUpper().Equals("QUIT")) break;
            data = new byte[1024];
            recv = client.ReceiveFrom(data, ref remote);
            s = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(s);
        }
        client.Close();
    }
}
```

Sử dụng Socket không hướng kết nối viết chương trình chat giưa 2 máy như sau: (Sau này chúng ta có thể sử dụng lớp UdpClient)

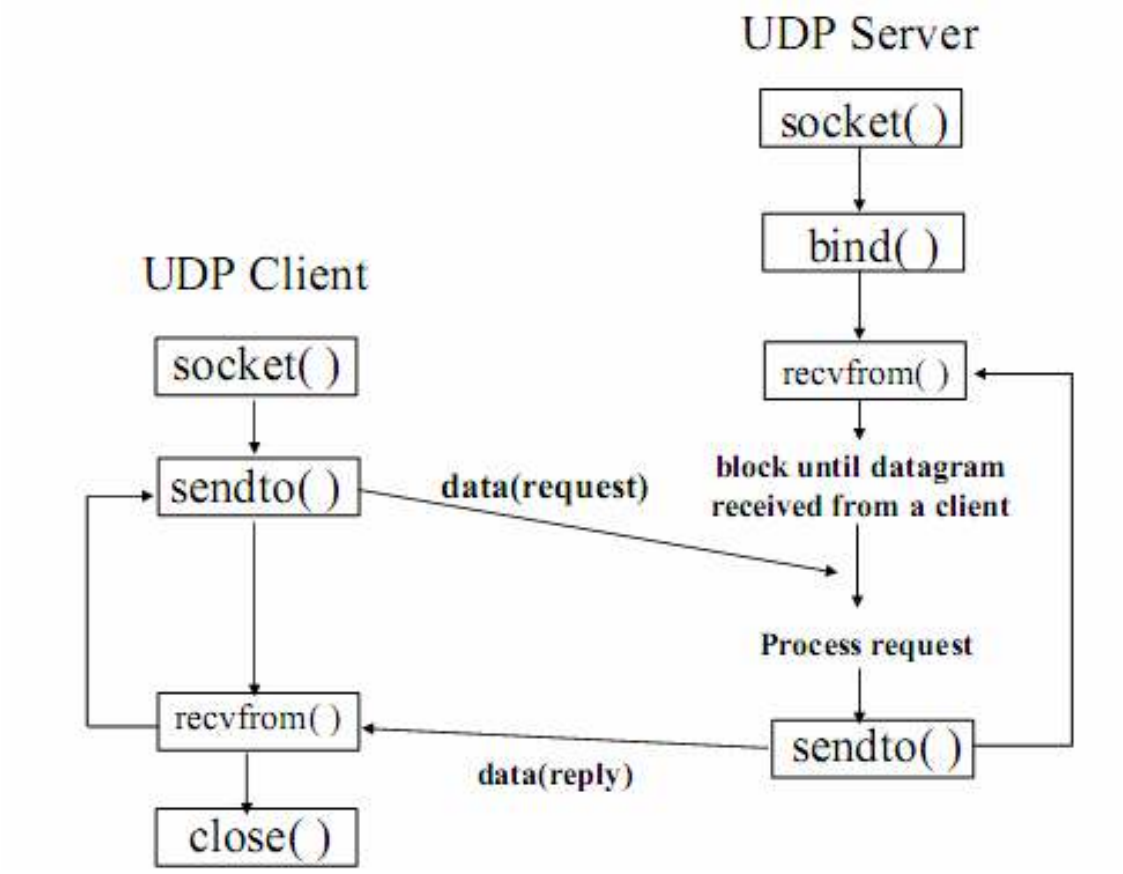
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;

public partial class Form1 : Form {
    private Socket udp1;
    private IPEndPoint ipremote, iplocal;
    public Form1() {
        InitializeComponent();
        CheckForIllegalCrossThreadCalls = false;
    }
    private void btStart_Click(object sender, EventArgs e) {
        udp1 = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
ProtocolType.Udp);
        iplocal = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
int.Parse(txtLocalPort.Text));
        udp1.Bind(iplocal);
        ipremote = new IPEndPoint(IPAddress.Parse(txtIp.Text),
int.Parse(txtRemotePort.Text));
        Thread tuyen = new Thread(new ThreadStart(NhanDL));
        tuyen.Start();
```

```
    }

    private void btSend_Click(object sender, EventArgs e) {
        byte[] data = new byte[1024];
        data = Encoding.ASCII.GetBytes(txtSend.Text);
        udp1.SendTo(data, ipremote);
    }
    private void NhanDL() {
        while (true) {
            byte[] data = new byte[1024];
            IPEndPoint ipe = new IPEndPoint(IPAddress.Any, 0);
            EndPoint remote = (EndPoint)ipe;
            int rec = udp1.ReceiveFrom(data, ref remote);
            string s = Encoding.ASCII.GetString(data, 0, rec);
            txtNoidung.Text += s + "\r\n";
        }
    }

    private void button1_Click(object sender, EventArgs e) {
        MessageBox.Show(txtSend.Text.Substring(0, txtSend.Text.IndexOf(" ")));
    }
}
```



2.2.3. Sử dụng lớp System.IO.MemoryStream để tạo vùng đệm nhập xuất

2.3. Sử dụng các lớp hỗ trợ được xây dựng từ lớp Soket

2.3.1. Lớp TCPClient

Mục đích của lớp UDPClient ở trên là dùng cho lập trình với giao thức UDP, với giao thức này thì hai bên không cần phải thiết lập kết nối trước khi gửi do vậy mức độ tin cậy không cao. Để đảm bảo độ tin cậy trong các ứng dụng mạng, người ta còn dùng một giao thức khác, gọi là giao thức có kết nối : TCP (Transport Control Protocol). Trên Internet chủ yếu là dùng loại giao thức này, ví dụ như Telnet, HTTP, SMTP, POP3… Để lập trình theo giao thức TCP, MS.NET cung cấp hai lớp có tên là TCPClient và TCPListener.

- Các thành phần của lớp TcpClient

+ Phương thức khởi tạo:

Constructor Method	
Name	**Description**
TcpClient ()	Tạo một đối tượng **TcpClient**. Chưa đặt thông số gì.
TcpClient (IPEndPoint)	Tạo một **TcpClient** và gắn cho nó một EndPoint cục bộ. (Gán địa chỉ máy cục bộ và số hiệu cổng để sử dụng trao đổi thông tin về sau)
TcpClient (RemoteHost: String, Int32)	Tạo một đối tượng **TcpClient** và kết nối đến một máy có địa chỉ và số hiệu cổng được truyền vào.. RemoteHost có thể là địa chỉ IP chuẩn hoặc tên máy.

+ Một số thuộc tính:

	Name	**Description**
	Available	Cho biết số byte đã nhận về từ mạng và có sẵn để đọc.
	Client	Trả về Socket ứng với TCPClient hiện hành.
	Connected	Trạng thái cho biết đã kết nối được đến Server hay chưa ?

+ Một số phương thức:

	Name	**Description**
	Close	Giải phóng đối tượng **TcpClient** nhưng không đóng kết nối.
	Connect (RemoteHost, Port)	Kết nối đến một máy TCP khác có Tên và số hiệu cổng.

	GetStream	Trả về NetworkStream để từ đó giúp ta gửi hay nhận dữ liệu. (Thường làm tham số khi tạo StreamReader và StreamWriter) . Khi đã gắn vào StreamReader và StreamWriter rồi thì ta có thể gửi và nhận dữ liệu thông qua các phương thức Readln, writeline tương ứng của các lớp này.

Ta sử dụng lớp TcpClient viết lại chương trình DateTimeClient như sau:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;
class DateTimeClient {
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9999);
      TcpClient client = new TcpClient();
      client.Connect(iep);
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      while (true) {
         string input = Console.ReadLine();
         sw.WriteLine(input);
         sw.Flush();
         if (input.ToUpper().Equals("QUIT")) break;
         string kq = sr.ReadLine();
         Console.WriteLine(kq);
      }
      sr.Close();
      sw.Close();
      client.Close();
   }
}
```

2.3.2. Lớp TCPListener

TCPListerner là một lớp cho phép người lập trình có thể xây dựng các ứng dụng Server (Ví dụ như SMTP Server, FTP Server, DNS Server, POP3 Server hay server tự định nghĩa ….). Ứng dụng server khác với ứng dụng Client ở chỗ nó luôn luôn thực hiện lắng nghe và chấp nhận các kết nối đến từ Client.

Các thành phần của lớp TcpListener:
+ Phương thức khởi tạo:

Constructor method	
Name	**Description**
TcpListener (Port: Int32)	Tạo một **TcpListener** và lắng nghe tại cổng chỉ định.
TcpListener (IPEndPoint)	Tạo một **TcpListener** với giá trị Endpoint truyền vào.
TcpListener (IPAddress, Int32)	Tạo một **TcpListener** và lắng nghe các kết nối đến tại địa chỉ IP và cổng chỉ định.

+ Các phương thức khác

	Name	Description
	AcceptSocket	Chấp nhận một yêu cầu kết nối đang chờ.
	AcceptTcpClient	Chấp nhận một yêu cầu kết nối đang chờ. (Ứng dụng sẽ dừng tại lệnh này cho đến khi nào có một kết nối đến)
	Pending	Cho biết liệu có kết nối nào đang chờ đợi không ? (True = có).
	Start	Bắt đầu lắng nghe các yêu cầu kết nối.
	Stop	Dừng việc nghe.

Sử dụng lớp TcpListener ta viết lại chương trình DateTime Server như sau:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
class DateTimeServer{
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2009);
      TcpListener server = new TcpListener(iep);
      server.Start();
      TcpClient client = server.AcceptTcpClient();
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      string kq="";
      while (true) {
```

```
            string s = sr.ReadLine();
            s=s.ToUpper();
            if (s.Equals("QUIT")) break;
            if (s.Equals("GETDATE"))
               kq = DateTime.Now.ToString("dd/MM/yyyy");
            else
               if (s.Equals("GETTIME"))
                  kq = DateTime.Now.ToString("hh:mm:ss");
               else
                  kq = "Khong hieu lenh";
            sw.WriteLine(kq);
            sw.Flush();
         }
         sr.Close();
         sw.Close();
         client.Close();
      }
   }
```

2.3.3. Lớp UDPClient

Giao thức UDP (User Datagram Protocol hay User Define Protocol) là một giao thức phi kết nối (Connectionless) có nghĩa là một bên có thể gửi dữ liệu cho bên kia mà không cần biết là bên đó đã sẵn sàng hay chưa ? (Nói cách khác là không cần thiết lập kết nối giữa hai bên khi tiến hành trao đổi thông tin). Giao thức này không tin cậy bằng giao thức TCP nhưng tốc độ lại nhanh và dễ cài đặt. Ngoài ra, với giao thức UDP ta còn có thể gửi các gói tin quảng bá (Broadcast) cho đồng thời nhiều máy.

Trong .NET, lớp **UDPClient** (nằm trong `System.Net.Sockets`) đóng gói các chức năng của giao thức UDP.

Constructor methosd	**Description**
UdpClient ()	Tạo một đối tượng (thể hiện) mới của lớp UDPClient.
UdpClient (AddressFamily)	Tạo một đối tượng (thể hiện) mới của lớp UDPClient. Thuộc một dòng địa chỉ (AddressFamily) được chỉ định.
UdpClient (Int32)	Tạo một **UdpClient** và gắn (bind) một cổng cho nó.
UdpClient (IPEndPoint)	Tạo một **UdpClient** và gắn (bind) một IPEndpoint (gán địa chỉ IP và cổng) cho nó.
UdpClient (Int32, AddressFamily)	Tạo một **UdpClient** và gán số hiệu cổng, AddressFamily

UdpClient (String, Int32)	Tạo một **UdpClient** và thiết lập với một trạm từ xa mặc định.

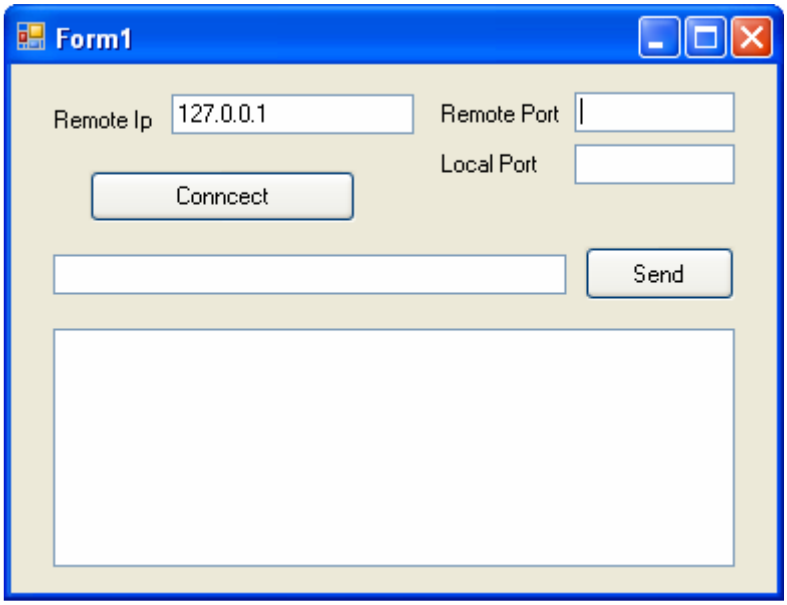
PUBLIC Method

	Name	Description
	BeginReceive	Nhận dữ liệu Không đồng bộ từ máy ở xa.
	BeginSend	Gửi không đồng bộ dữ liệu tới máy ở xa
	Close	Đóng kết nối.
	Connect	Thiết lập một Default remote host.
	EndReceive	Kết thúc nhận dữ liệu không đồng bộ ở trên
	EndSend	Kết thúc việc gửi dữ liệu không đồng bộ ở trên
	Receive	**Nhận dữ liệu (đồng bộ) do máy ở xa gửi.** (Đồng bộ có nghĩa là các lệnh ngay sau lệnh Receive chỉ được thực thi nếu Receive đã nhận được dữ liệu về . Còn nếu nó chưa nhận được – dù chỉ một chút – thì nó vẫn cứ chờ (blocking))
	Send	**Gửi dữ liệu (đồng bộ) cho máy ở xa.**

Ví dụ sử dụng UdpClient viết chương trình Chat giữa 2 máy:

Do chương trình ở 2 máy là như nhau ta chỉ cần viết một chương trình copy ra để sử dụng.

Hình ảnh của nó như sau:



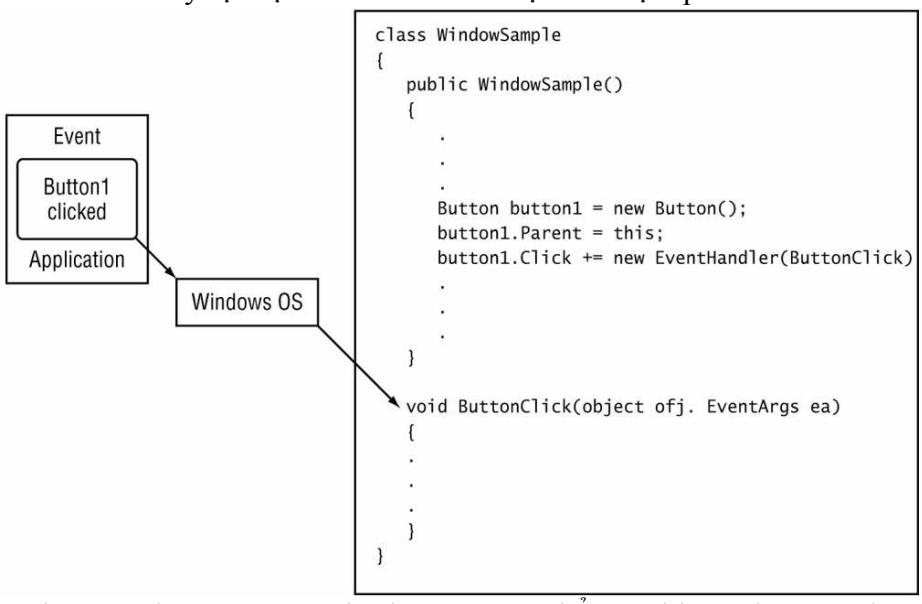

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
namespace UdpChat {
   public partial class Form1 : Form {
      public Form1() {
         InitializeComponent();
         CheckForIllegalCrossThreadCalls = false;
      }
      private void btSend_Click(object sender, EventArgs e) {
         UdpClient send = new UdpClient();
         IPEndPoint iepRemote = new IPEndPoint(IPAddress.Parse(txtIp.Text),
int.Parse(txtRemotePort.Text));
         byte[] data = new byte[1024];
         data = Encoding.UTF8.GetBytes(txtSend.Text);
         send.Send(data, data.Length, iepRemote);
         txtReceive.Text += "Sender: "+txtSend.Text + "\r\n";
         txtSend.Clear();
         if (txtSend.Text.ToUpper().Equals("QUIT")) this.Dispose();
      }
      private void btConnect_Click(object sender, EventArgs e) {
         Thread tuyen = new Thread(new ThreadStart(NhanDl));
         tuyen.Start();
      }
      private void NhanDl() {
         UdpClient receiver = new UdpClient(int.Parse(txtLocalPort.Text));
         IPEndPoint iep = new IPEndPoint(IPAddress.Any, 0);
         while (true) {
            byte[] data = new byte[1024];
            data = receiver.Receive(ref iep);
            string s = Encoding.UTF8.GetString(data);
            if (s.Trim().ToUpper().Equals("QUIT")) break;
            txtReceive.Text += "Receiver: "+ s + "\r\n";
         }
      }
   }
}
```

2.4. Socket không đồng bộ

2.4.1. Mô hình xử lý sự kiện của windows

Mô hình xử lý sự kiện của Windows được thể hiện qua hình sau:



Như vậy thông qua mô hình này ta có thể ủy nhiệm cho một thủ tục nào đó thực hiện khi sự kiện sảy ra trên các Control

Ví dụ:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace EventDemo {
   public partial class Form1 : Form {
      public Form1() {
         InitializeComponent();
         button1.Click += new EventHandler(NhanTiep);
      }

      private void button1_Click(object sender, EventArgs e) {
         MessageBox.Show("Bac da nhan em");
      }
      private void NhanTiep(object sender, EventArgs e) {
         MessageBox.Show("Bac lai nhan em roi");
      }
   }
}
```

Ở ví dụ trên chúng ta ngoài sự kiện Click của button 1 chúng ta thêm một xự kiện khi button1 được nhấn đó là sự kiện NhanTiep.

2.4.2. Sử dụng Socket không đồng bộ

Để lập trình không đồng bộ với Socket chúng ta sử dụng các phương thức cho việc sử dụng bất đồng bộ

Các phương thức cho việc lập trình bất đồng bộ được chia làm 2 lại thường bắt đầu bằng Begin và End:

- Phương thức bắt đầu bằng Begin, bắt đầu một chức năng và được đăng ký với phương thức AsyncCallback
- Bắt đầu bằng End chỉ chức năng hoàn thành khi AsyncCallback được gọi.

Requests Started By…	Description of Request	Requests Ended BY…
BeginAccept()	To accept an incoming connection	EndAccept()
BeginConnect()	To connect to a remote host	EndConnect()
BeginReceive()	To retrieve data from a socket	EndReceive()
BeginReceiveFrom()	To retrieve data from a specific remote host	EndReceiveFrom()
BeginSend()	To send data from a socket	EndSend()
BeginSendTo()	To send data to a specific remote host	EndSendTo()

- Để chấp nhận kết nối bất đồng bộ ta sử dụng phương thức BeginAccept() và EndAccept() như sau:

- Phương thức BeginAccept() và EndAccept()

  IAsyncResult BeginAccept(AsyncCallback callback, object state)

  Socket EndAccept(IAsyncResult iar);

- Thường được dùng như sau:

```
Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                                                        ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
sock.Bind(iep);
sock.Listen(5);
sock.BeginAccept(new AsyncCallback(CallAccept), sock);
```

Trong đó phương thức CallAccept thường được viết như sau:

```
    private static void CallAccept(IAsyncResult iar) {
                Socket server = (Socket)iar.AsyncState;
                Socket client = server.EndAccept(iar);
                 . . .
    }
```

- Để thiết lập kết nối theo cách bất đồng bộ chúng ta sử dụng phương thức BeginConnect() và EndConnect() như sau:

- Phương thức BeginConnect() và EndConnect()

Socket newsock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

IPEndPoint iep =new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);

newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);

Trong đó phương thức Connected thường được viết như sau:

```
public static void Connected(IAsyncResult iar) {
        Socket sock = (Socket)iar.AsyncState;
        try {
                sock.EndConnect(iar);
        } catch (SocketException) {
                Console.WriteLine("Unable to connect to host");
        }
}
```

- Để gửi dữ liệu bất đồng bộ chúng ta làm như sau:

+ Phương thức BeginSend() và EndSend()

+ BeginSend()

IAsyncResult BeginSend(byte[] *buffer*, int *offset*, int *size*, SocketFlags *sockflag*, AsyncCallback *callback*, object *state*)

Ví dụ:

sock.BeginSend(data, 0, data.Length, SocketFlags.None, new AsyncCallback(SendData), sock);

+ EndSend()

int EndSend(IAsyncResult iar)

Trong đó phương thức SendData thường được viết như sau:

```
private static void SendData(IAsyncResult iar) {
        Socket server = (Socket)iar.AsyncState;
        int sent = server.EndSend(iar);
}
```

Tương tự như giao thức hướng kết nối nếu ta sử dụng gửi dữ liệu theo giao thức không hướng kết nối chúng ta cũng thực hiện tương tự như sau:

- ❑ Phương thức BeginSendTo() và EndSendTo()

IAsyncResult BeginSendTo(byte[] buffer, int offset, int size, SocketFlags sockflag, EndPoint ep, AsyncCallback callback, object state)

Ví dụ:

IPEndPoint iep = new EndPoint(IPAddress.Parse("192.168.1.6"), 9050);

sock.BeginSendTo(data, 0, data.Length, SocketFlags.None, iep, new AsynCallback(SendDataTo), sock);

int EndSendTo(IAsyncResult iar)

- Để nhận dữ liệu bất đồng bộ ta thực hiện như sau:

+ Nhận dữ liệu với giao thức hướng kết nối:

- ❑ Phương thức BeginReceive và EndReceive()

```
sock.BeginReceive(data, 0, data.Length, SocketFlags.None, new
AsyncCallback(ReceivedData), sock);
```

Với ReceivedData được định nghĩa như sau:

```
void ReceivedData(IAsyncResult iar) {
        Socket remote = (Socket)iar.AsyncState;
        int recv = remote.EndReceive(iar);
        string receivedData = Encoding.ASCII.GetString(data, 0,
    recv);
        Console.WriteLine(receivedData);
}
```

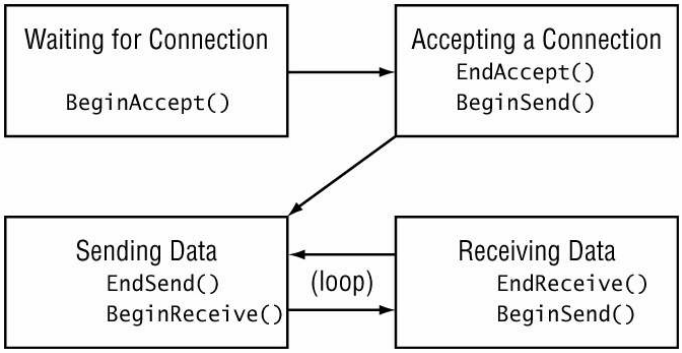
+ Nhận dữ liệu bất đồng bộ với giao thức không hướng kết nối.

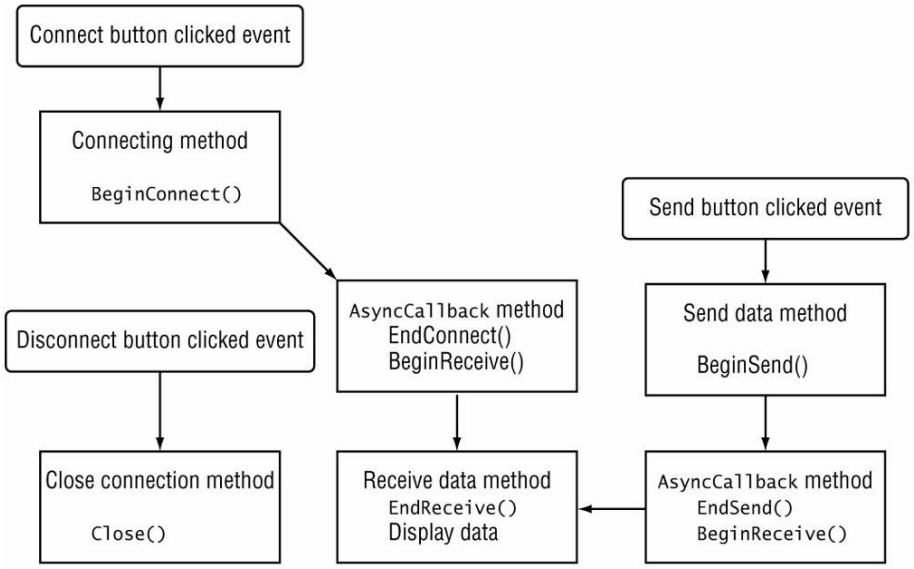
- Phương thức BeginReceiveFrom() and EndReceiveFrom()

```
sock.BeginReceive(data, 0, data.Length, SocketFlags.None, ref iep, new
AsyncCallback(ReceiveData), sock);
void ReceiveData(IasyncResult iar){
        Socket remote = (Socket)iar.AsyncState;
        int recv = remote.EndReceiveFrom(iar);
        string stringData = Encoding.ASCII.GetString(data, 0,
recv);
        Console.WriteLine(stringData);
}
```

2.4.3. Ví dụ về Socket không đồng bộ

Sau đây chúng ta sẽ sử dụng các phương thức không đồng bộ viết chương trình Client/Server theo Socket bất đồng bộ, mỗi khi Client gửi dữ liệu lên Server, nó sẽ in ra và gửi trả lại cho Client. Mô hình của client/server sử dụng các phương thức bất đồng bộ như sau:





Chương trình phía Client:

```
using System;
using System.Drawing;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Windows.Forms;
class AsyncTcpClient:Form
{
 private TextBox newText;
 private TextBox conStatus;
 private ListBox results;
 private Socket client;
 private byte[] data = new byte[1024];
 private int size = 1024;
 public AsyncTcpClient()
 {
  Text = "Asynchronous TCP Client";
  Size = new Size(400, 380);
  Label label1 = new Label();
  label1.Parent = this;
  label1.Text = "Enter text string:";
  label1.AutoSize = true;
  label1.Location = new Point(10, 30);
  newText = new TextBox();
  newText.Parent = this;
  newText.Size = new Size(200, 2 * Font.Height);
  newText.Location = new Point(10, 55);
  results = new ListBox();
  results.Parent = this;
  results.Location = new Point(10, 85);
  results.Size = new Size(360, 18 * Font.Height);
  Label label2 = new Label();
  label2.Parent = this;
```

```
 label2.Text = "Connection Status:";
 label2.AutoSize = true;
 label2.Location = new Point(10, 330);
 conStatus = new TextBox();
 conStatus.Parent = this;
 conStatus.Text = "Disconnected";
 conStatus.Size = new Size(200, 2 * Font.Height);
 conStatus.Location = new Point(110, 325);
 Button sendit = new Button();
 sendit.Parent = this;
 sendit.Text = "Send";
 sendit.Location = new Point(220,52);
 sendit.Size = new Size(5 * Font.Height, 2 * Font.Height);
 sendit.Click += new EventHandler(ButtonSendOnClick);
 Button connect = new Button();
 connect.Parent = this;
 connect.Text = "Connect";
 connect.Location = new Point(295, 20);
 connect.Size = new Size(6 * Font.Height, 2 * Font.Height);
 connect.Click += new EventHandler(ButtonConnectOnClick);
 Button discon = new Button();
 discon.Parent = this;
 discon.Text = "Disconnect";
 discon.Location = new Point(295,52);
 discon.Size = new Size(6 * Font.Height, 2 * Font.Height);
 discon.Click += new EventHandler(ButtonDisconOnClick);
}
void ButtonConnectOnClick(object obj, EventArgs ea)
{
 conStatus.Text = "Connecting...";
 Socket newsock = new Socket(AddressFamily.InterNetwork,
          SocketType.Stream, ProtocolType.Tcp);
 IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
 newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);
}
void ButtonSendOnClick(object obj, EventArgs ea)
{
 byte[] message = Encoding.ASCII.GetBytes(newText.Text);
 newText.Clear();
 client.BeginSend(message, 0, message.Length, SocketFlags.None,
       new AsyncCallback(SendData), client);
}
void ButtonDisconOnClick(object obj, EventArgs ea)
{
 client.Close();
 conStatus.Text = "Disconnected";
}
void Connected(IAsyncResult iar)
{
```

```
  client = (Socket)iar.AsyncState;
  try
  {
   client.EndConnect(iar);
   conStatus.Text = "Connected to: " + client.RemoteEndPoint.ToString();
   client.BeginReceive(data, 0, size, SocketFlags.None,
        new AsyncCallback(ReceiveData), client);
  } catch (SocketException)
  {
   conStatus.Text = "Error connecting";
  }
 }
 void ReceiveData(IAsyncResult iar)
 {
  Socket remote = (Socket)iar.AsyncState;
  int recv = remote.EndReceive(iar);
  string stringData = Encoding.ASCII.GetString(data, 0, recv);
  results.Items.Add(stringData);
 }
 void SendData(IAsyncResult iar)
 {
  Socket remote = (Socket)iar.AsyncState;
  int sent = remote.EndSend(iar);
  remote.BeginReceive(data, 0, size, SocketFlags.None,
       new AsyncCallback(ReceiveData), remote);
 }
 public static void Main()
 {
  Application.Run(new AsyncTcpClient());
 }
}
```

Chương trình phía Server:

```
using System;
using System.Drawing;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Windows.Forms;
class AsyncTcpSrvr    :Form
{
 private TextBox conStatus;
 private ListBox results;
 private byte[] data = new byte[1024];
 private int size = 1024;
 private Socket server;
 public AsyncTcpSrvr()
 {
  Text = "Asynchronous TCP Server";
  Size = new Size(400, 380);
```

```
 results = new ListBox();
 results.Parent = this;
 results.Location = new Point(10, 65);
 results.Size = new Size(350, 20 * Font.Height);
 Label label1 = new Label();
 label1.Parent = this;
 label1.Text = "Text received from client:";
 label1.AutoSize = true;
 label1.Location = new Point(10, 45);
 Label label2 = new Label();
 label2.Parent = this;
 label2.Text = "Connection Status:";
 label2.AutoSize = true;
 label2.Location = new Point(10, 330);
 conStatus = new TextBox();
 conStatus.Parent = this;
 conStatus.Text = "Waiting for client...";
 conStatus.Size = new Size(200, 2 * Font.Height);
 conStatus.Location = new Point(110, 325);
 Button stopServer = new Button();
 stopServer.Parent = this;
 stopServer.Text = "Stop Server";
 stopServer.Location = new Point(260,32);
 stopServer.Size = new Size(7 * Font.Height, 2 * Font.Height);
 stopServer.Click += new EventHandler(ButtonStopOnClick);
 server = new Socket(AddressFamily.InterNetwork,
       SocketType.Stream, ProtocolType.Tcp);
 IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
 server.Bind(iep);
 server.Listen(5);
 server.BeginAccept(new AsyncCallback(AcceptConn), server);
}
void ButtonStopOnClick(object obj, EventArgs ea)
{
 Close();
}
void AcceptConn(IAsyncResult iar)
{
 Socket oldserver = (Socket)iar.AsyncState;
 Socket client = oldserver.EndAccept(iar);
 conStatus.Text = "Connected to: " + client.RemoteEndPoint.ToString();
 string stringData = "Welcome to my server";
 byte[] message1 = Encoding.ASCII.GetBytes(stringData);
 client.BeginSend(message1, 0, message1.Length, SocketFlags.None,
      new AsyncCallback(SendData), client);
}
void SendData(IAsyncResult iar)
{
 Socket client = (Socket)iar.AsyncState;
```

```
  int sent = client.EndSend(iar);
  client.BeginReceive(data, 0, size, SocketFlags.None,
       new AsyncCallback(ReceiveData), client);
 }
 void ReceiveData(IAsyncResult iar)
 {
  Socket client = (Socket)iar.AsyncState;
  int recv = client.EndReceive(iar);
  if (recv == 0)
  {
   client.Close();
   conStatus.Text = "Waiting for client...";
   server.BeginAccept(new AsyncCallback(AcceptConn), server);
   return;
  }
  string receivedData = Encoding.ASCII.GetString(data, 0, recv);
  results.Items.Add(receivedData);
  byte[] message2 = Encoding.ASCII.GetBytes(receivedData);
  client.BeginSend(message2, 0, message2.Length, SocketFlags.None,
       new AsyncCallback(SendData), client);
 }
 public static void Main()
 {
  Application.Run(new AsyncTcpSrvr());
 }
}
```

2.4.4. Sử dụng các phương thức Non-blocking

Để lập trình bất đồng bộ chúng ta có thể sử dụng các phương thức Non – bloking như phương thức Poll() và phương thức Select:

+ Phương thức Poll()

bool Poll(int microseconds, SelectMode mode);

- SelectRead: Poll() trả về true nếu một trong những điều kiện sau được thoả:
  - Nếu phương thức Accept() thành công
  - Nếu có dữ liệu trên Socket
  - Nếu kết nối đã đóng
- SelectWrite: Poll() trả về true nếu thoả một trong những điều kiện sau:
  - Phương thức Connect() thành công
  - Nếu có dữ liệu trên Socket để gửi
- SelectError: Poll() trả về true nếu một trong những điều kiện sau được thoả:
  - Nếu phương thức Connect() thất bại
  - Nếu có dữ liệu ngoài băng thông chuẩn gửi đến nhưng thuộc tính OutOfBandInline không được thiết lập là true.

+ Phương thức Select():

Socket.Select(IList checkRead, IList checkWrite, IList checkError, int microseconds)

• *checkRead* monitors the specified sockets for the ability to read data from the socket.

• *checkWrite* monitors the specified sockets for the ability to write data to the socket.

• *checkError* monitors the specified sockets for error conditions.

Ví dụ ứng dụng Server sử dụng phương thức Poll()

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpPollSrvr
{
   public static void Main()
   {
      int recv;
      byte[] data = new byte[1024];
      IPEndPoint ipep = new IPEndPoint(IPAddress.Any,
               9050);
      Socket newsock = new
        Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
      newsock.Bind(ipep);
      newsock.Listen(10);
      Console.WriteLine("Waiting for a client...");
      bool result;
      int i = 0;
      while (true)
      {
         i++;
         Console.WriteLine("polling for accept#{0}...", i);
         result = newsock.Poll(1000000, SelectMode.SelectRead);
         if (result)
         {
            break;
         }
      }
      Socket client = newsock.Accept();
      IPEndPoint newclient =
           (IPEndPoint)client.RemoteEndPoint;
      Console.WriteLine("Connected with {0} at port {1}",
            newclient.Address, newclient.Port);

      string welcome = "Welcome to my test server";
      data = Encoding.ASCII.GetBytes(welcome);
      client.Send(data, data.Length,
             SocketFlags.None);
```

```
        i = 0;
        while (true)
        {
           Console.WriteLine("polling for receive #{0}...", i);
           i++;
           result = client.Poll(3000000, SelectMode.SelectRead);
           if (result)
           {
              data = new byte[1024];
              i = 0;
              recv = client.Receive(data);
              if (recv == 0)
                 break;

              Console.WriteLine(
                Encoding.ASCII.GetString(data, 0, recv));
              client.Send(data, recv, 0);
           }
        }
        Console.WriteLine("Disconnected from {0}",
              newclient.Address);
        client.Close();
        newsock.Close();
     }
}
```

Sau đây chúng ta sẽ viết một chương trình Server sử dụng phương thức Select() để chấp nhận 2 kết nối đến từ client và xử lý từng kết nối.

Chương trình Select Server:

```
using System;
using System.Collections;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SelectTcpSrvr
{
   public static void Main()
   {
      ArrayList sockList = new ArrayList(2);
      ArrayList copyList = new ArrayList(2);
      Socket main = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
      byte[] data = new byte[1024];
      string stringData;
```

```
int recv;
main.Bind(iep);
main.Listen(2);
Console.WriteLine("Waiting for 2 clients...");
Socket client1 = main.Accept();
IPEndPoint iep1 = (IPEndPoint)client1.RemoteEndPoint;
client1.Send(Encoding.ASCII.GetBytes("Welcome to my server"));
Console.WriteLine("Connected to {0}", iep1.ToString());
sockList.Add(client1);
Console.WriteLine("Waiting for 1 more client...");
Socket client2 = main.Accept();
IPEndPoint iep2 = (IPEndPoint)client2.RemoteEndPoint;
client2.Send(Encoding.ASCII.GetBytes("Welcome to my server"));
Console.WriteLine("Connected to {0}", iep2.ToString());
sockList.Add(client2);
main.Close();
while (true)
{
   copyList = new ArrayList(sockList);
   Console.WriteLine("Monitoring {0} sockets...", copyList.Count);
   Socket.Select(copyList, null, null, 10000000);
   foreach (Socket client in copyList)
   {
      data = new byte[1024];
      recv = client.Receive(data);
      stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine("Received: {0}", stringData);
      if (recv == 0)
      {
         iep = (IPEndPoint)client.RemoteEndPoint;
         Console.WriteLine("Client {0} disconnected.", iep.ToString());
         client.Close();
         sockList.Remove(client);
         if (sockList.Count == 0)
         {
            Console.WriteLine("Last client disconnected, bye");
            return;
         }
      }
      else
         client.Send(data, recv, SocketFlags.None);
```

```
            }
        }
    }
}
```

Chương trình Client:

```
using System;
using System.Collections;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SelectTcpClient
{
    public static void Main()
    {
        Socket sock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
        byte[] data = new byte[1024];
        string stringData;
        int recv;
        sock.Connect(iep);
        Console.WriteLine("Connected to server");
        recv = sock.Receive(data);
        stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine("Received: {0}", stringData);
        while (true)
        {
            stringData = Console.ReadLine();
            if (stringData == "exit")
                break;
            data = Encoding.ASCII.GetBytes(stringData);
            sock.Send(data, data.Length, SocketFlags.None);
            data = new byte[1024];
            recv = sock.Receive(data);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine("Received: {0}", stringData);
        }
        sock.Close();
    }
}
```

2.5. Sử dụng Thread trong các ứng dụng mạng

Một số khái niệm

- Đa nhiệm (Multitasking): Là khả năng hệ điêu hành làm nhiều công việc tại một thời điểm
- Tiến trình (Process): Khi chạy một ứng dụng, hệ điều hành sẽ cấp phát riêng cho ứng dụng đó bộ nhớ và các tài nguyên khác. Bộ nhớ và tài nguyên vật lý riêng biệt này được gọi là một tiến trình. Các tài nguyên và bộ nhớ của một tiến trình thì chỉ tiến trình đó được phép truy cập.
- Tuyến (Thread): Trong hệ thống, một tiến trình có thể có một hoặc nhiều chuỗi thực hiện tách biệt nhau và có thể chạy đồng thời. Mỗi chuỗi thực hiện này được gọi là một tuyến (Thread). Trong một ứng dụng, Thread khởi tạo đầu tiên gọi là Thread sơ cấp hay Thread chính.

2.5.1. Sử dụng Thread trong chương trình .Net

Để sử dụng Thread trong .Net ta sử dụng NameSpace System.Threading

- Một số phương thức thường dùng

Public Method Name	**Mô tả**
Abort()	Kết thúc Thread
Join()	Buộc chương trình phải chờ cho thread kết thúc (Block) thì mới thực hiện tiếp (các câu lệnh đứng sau Join).
Resume()	Tiếp tục chạy thread đã bị tạm ngưng - suspended.
Sleep()	Static method : Tạm dừng thread trong một khoảng thời gian.
Start()	Bắt đầu chạy (khởi động) một thread. Sau khi gọi phương thức này, trạng thái của thread chuyển từ trạng thái hiện hành sang Running.
Suspend()	Tạm ngưng (nghỉ) thread. (Phương thức này đã bị loại khỏi phiên bản VS.NET 2005)

- Một số thuộc tính thường dùng:

Public Property Name	**Mô tả**
CurrentThread	This static property: Trả về thread hiện hành đang chạy.
IsAlive	Trả về giá trị cho biết trạng thái thực thi của thread hiện hành.
IsBackground	Sets or gets giá trị cho biết là thread là background hay foreground thread.
IsThreadPoolThread	Gets a value indicating whether a thread is part of a thread pool.
Priority	Sets or gets giá trị để chỉ định độ ưu tiên (dành nhiều hay ít CPU cho thread). Cao nhất là 4, thấp nhất là 0.

Public Property Name	Mô tả
ThreadState	Lấy về trạng thái của thread (đang dừng, hay đang chạy…)

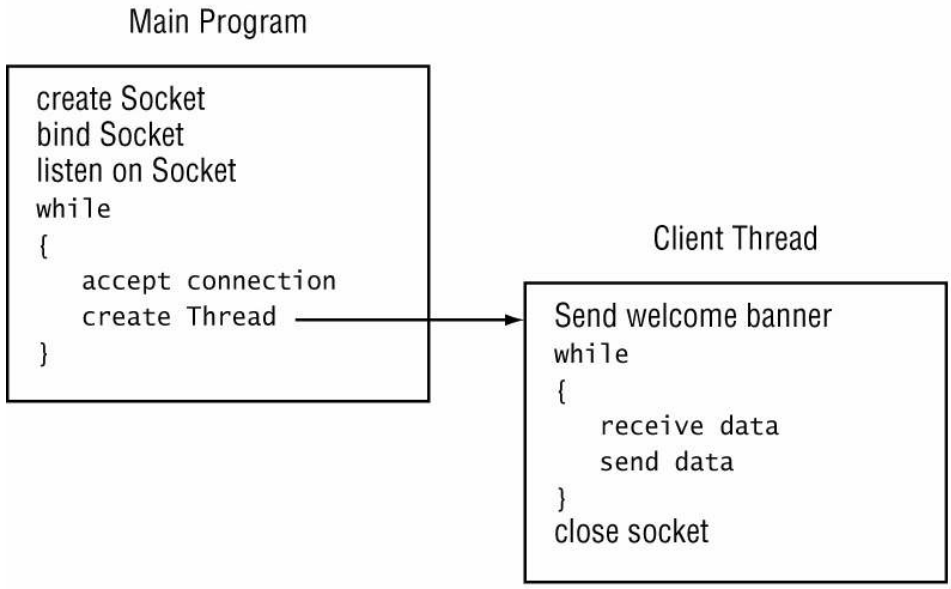
- Tạo một tuyến trong C#

```
……
Thread newThread=newThread(new ThreadStart(newMethod));
…….
 }
void newMethod() {
. . .
}
```

2.5.2. Sử dụng Thread trong các chương trình Server

- Đa tuyên hay được ứng dụng trong các chương trình Server, các chương trình đòi hỏi tại một thời điểm chấp nhận nhiều kết nối đến từ các Client.

- Để các chương trình Server có thể xử lý nhiều Client tại một thời điểm ta có mô hình ứng dụng đa tuyến như sau:



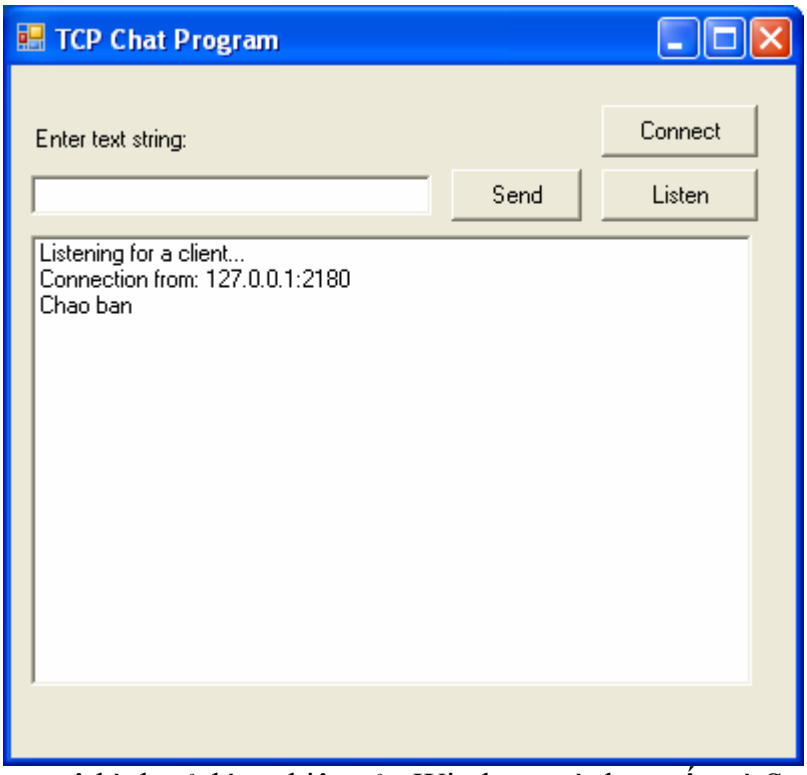
Sau đây chúng ta viết lại chương trình DateTimeServer có sử dụng Thread như sau:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.IO;
class Program {
   static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2009);
      TcpListener server = new TcpListener(iep);
      server.Start();
      while (true) {
```

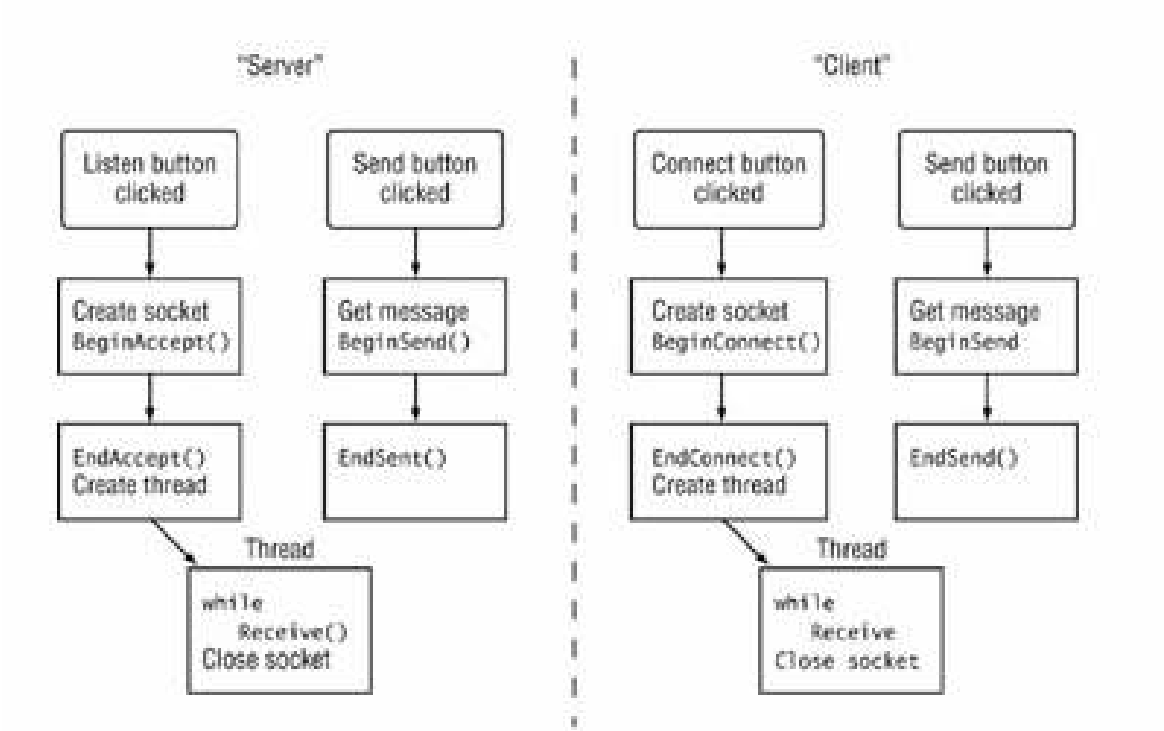
```
            //chap nhan ket noi
            TcpClient client= server.AcceptTcpClient();
            //Tao ra tuyen moi de xu ly moi Client
            new ClientThread(client);
        }
        server.Stop();
    }
}
class ClientThread {
    private Thread tuyen;
    private TcpClient client;
    public ClientThread(TcpClient client) {
        this.client = client;
        tuyen = new Thread(new ThreadStart(GuiNhanDL));
        tuyen.Start();
    }
    private void GuiNhanDL() {
        StreamReader sr = new StreamReader(client.GetStream());
        StreamWriter sw= new StreamWriter(client.GetStream());
        string kq="";
        while(true)
        {
            string s=sr.ReadLine();
            s=s.ToUpper();
            if(s.Equals("QUIT")) break;
            if(s.Equals("GETDATE"))
                kq=DateTime.Now.ToString("dd/MM/yyyy");
            else
                if(s.Equals("GETTIME"))
                    kq=DateTime.Now.ToString("hh:mm:ss");
                else
                    kq="Khong hieu lenh";
            sw.WriteLine(kq);
            sw.Flush();
        }
        client.Close();
    }
}
```

2.5.3. Sử dụng Thread để gửi/nhận dữ liệu

Ứng dụng đa tuyến trong việc gửi nhận dữ liệu chúng ta viết chương trình Chat theo giao thức TCP như sau:



Ứng dụng mô hình xử lý sự kiện của Windows và đa tuyến và Socket không đồng bộ ta chỉ cần viết một chương trình sau đó dịch ra, ta chạy ứng dụng nhấn Listen nó sẽ lắng nghe trong vai trò Server còn khi ta chạy và nhấn Connect nó sẽ đóng vai trò Client và kết nối tới Server.



Văn bản chương trình như sau:

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

```
using System.Windows.Forms;
class TcpChat:Form
{
 private static TextBox newText;
 private static ListBox results;
 private static Socket client;
 private static byte[] data = new byte[1024];
 public TcpChat()
 {
  Text = "TCP Chat Program";
  Size = new Size(400, 380);

  Label label1 = new Label();
  label1.Parent = this;
  label1.Text = "Enter text string:";
  label1.AutoSize = true;
  label1.Location = new Point(10, 30);
  newText = new TextBox();
  newText.Parent = this;
  newText.Size = new Size(200, 2 * Font.Height);
  newText.Location = new Point(10, 55);
  results = new ListBox();
  results.Parent = this;
  results.Location = new Point(10, 85);
  results.Size = new Size(360, 18 * Font.Height);
  Button sendit = new Button();
  sendit.Parent = this;
  sendit.Text = "Send";
  sendit.Location = new Point(220,52);
  sendit.Size = new Size(5 * Font.Height, 2 * Font.Height);
  sendit.Click += new EventHandler(ButtonSendOnClick);
  Button connect = new Button();
  connect.Parent = this;
  connect.Text = "Connect";
  connect.Location = new Point(295, 20);
  connect.Size = new Size(6 * Font.Height, 2 * Font.Height);
  connect.Click += new EventHandler(ButtonConnectOnClick);
  Button listen = new Button();
  listen.Parent = this;
  listen.Text = "Listen";
  listen.Location = new Point(295,52);
  listen.Size = new Size(6 * Font.Height, 2 * Font.Height);
  listen.Click += new EventHandler(ButtonListenOnClick);
 }
 void ButtonListenOnClick(object obj, EventArgs ea)
 {
  results.Items.Add("Listening for a client...");
  Socket newsock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
   ProtocolType.Tcp);
```

```
 IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
 newsock.Bind(iep);
 newsock.Listen(5);
 newsock.BeginAccept(new AsyncCallback(AcceptConn), newsock);
}
void ButtonConnectOnClick(object obj, EventArgs ea)
{
 results.Items.Add("Connecting...");
 client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
  ProtocolType.Tcp);
 IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
 client.BeginConnect(iep, new AsyncCallback(Connected), client);
}
void ButtonSendOnClick(object obj, EventArgs ea)
{
 byte[] message = Encoding.ASCII.GetBytes(newText.Text);
 newText.Clear();
 client.BeginSend(message, 0, message.Length, 0,
  new AsyncCallback(SendData), client);
}
void AcceptConn(IAsyncResult iar)
{
 Socket oldserver = (Socket)iar.AsyncState;
 client = oldserver.EndAccept(iar);
 results.Items.Add("Connection from: " + client.RemoteEndPoint.ToString());
 Thread receiver = new Thread(new ThreadStart(ReceiveData));
 receiver.Start();
}
void Connected(IAsyncResult iar)
{
 try
 {
  client.EndConnect(iar);
  results.Items.Add("Connected to: " + client.RemoteEndPoint.ToString());
  Thread receiver = new Thread(new ThreadStart(ReceiveData));
  receiver.Start();
 } catch (SocketException)
 {
  results.Items.Add("Error connecting");
 }
}
void SendData(IAsyncResult iar)
{
 Socket remote = (Socket)iar.AsyncState;
 int sent = remote.EndSend(iar);
}
void ReceiveData()
{
 int recv;
```

```
  string stringData;
  while (true)
  {
    recv = client.Receive(data);
    stringData = Encoding.ASCII.GetString(data, 0, recv);
    if (stringData == "bye")
     break;
    results.Items.Add(stringData);
  }
  stringData = "bye";
  byte[] message = Encoding.ASCII.GetBytes(stringData);
  client.Send(message);
  client.Close();
  results.Items.Add("Connection stopped");
  return;
 }
 public static void Main()
 {
  Application.Run(new TcpChat());
 }
}
```

2.5.4. Sử dụng ThreadPool trong các chương trình .Net

Method	**Description**
BindHandle()	Binds an operating system handle to the thread pool
GetAvailableThreads()	Gets the number of worker threads available for use in the thread pool
GetMaxThreads()	Gets the maximum number of worker threads available in the thread pool
QueueUserWorkItem()	Queues a user delegate to the thread pool
RegisterWaitForSingleObject()	Registers a delegate waiting for a WaitHandle object
UnsafeQueueUserWorkItem()	Queues an unsafe user delegate to the thread pool but does not propagate the calling stack onto the worker thread
UnsafeRegisterWaitForSingleObject()	Registers an unsafe delegate waiting for a WaitHandle object

```
using System;
using System.Threading;
class ThreadPoolSample
{
```

```
  public static void Main()
  {
   ThreadPoolSample tps = new ThreadPoolSample();
  }
  public ThreadPoolSample()
  {
   int i;
   ThreadPool.QueueUserWorkItem(new WaitCallback(Counter));
   ThreadPool.QueueUserWorkItem(new WaitCallback(Counter2));
   for(i = 0; i < 10; i++)
   {
     Console.WriteLine("main: {0}", i);
     Thread.Sleep(1000);
   }
  }
  void Counter(object state)
  {
   int i;
   for (i = 0; i < 10; i++)
   {
     Console.WriteLine(" thread: {0}", i);
     Thread.Sleep(2000);
   }
  }
  void Counter2(object state)
  {
   int i;
   for (i = 0; i < 10; i++)
   {
     Console.WriteLine("  thread2: {0}", i);
     Thread.Sleep(3000);
   }
  }
}
```

2.5.5. Sử dụng ThreadPool trong các chương trình Server

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
class ThreadPoolTcpSrvr
{
  private TcpListener client;
  public ThreadPoolTcpSrvr()
  {
   client = new TcpListener(9050);
   client.Start();
   Console.WriteLine("Waiting for clients...");
   while(true)
   {
     while (!client.Pending())
     {
      Thread.Sleep(1000);
     }
     ConnectionThread newconnection = new ConnectionThread();
     newconnection.threadListener = this.client;
     ThreadPool.QueueUserWorkItem(new
          WaitCallback(newconnection.HandleConnection));
   }
  }
  public static void Main()
  {
   ThreadPoolTcpSrvr tpts = new ThreadPoolTcpSrvr();
```

```
  }
}
class ConnectionThread
{
  public TcpListener threadListener;
  private static int connections = 0;
  public void HandleConnection(object state)
  {
   int recv;
   byte[] data = new byte[1024];
   TcpClient client = threadListener.AcceptTcpClient();
   NetworkStream ns = client.GetStream();
   connections++;
   Console.WriteLine("New client accepted: {0} active connections",
             connections);
   string welcome = "Welcome to my test server";
   data = Encoding.ASCII.GetBytes(welcome);
   ns.Write(data, 0, data.Length);
   while(true)
   {
     data = new byte[1024];
     recv = ns.Read(data, 0, data.Length);
     if (recv == 0)
      break;

     ns.Write(data, 0, recv);
   }
   ns.Close();
   client.Close();
   connections—;
   Console.WriteLine("Client disconnected: {0} active connections",
             connections);
  }
}
```

2.6. Kỹ thuật IP Multicasting

2.6.1. Broadcasting là gì?

Broadcast, tiếng Việt gọi là quảng bá. Trong hệ thống mạng hữu tuyến, quảng bá là thuật ngữ dùng để chỉ việc gửi một gói thông tin đến tất các nút mạng trong mạng. Để thực hiện hình thức quảng bá, địa chỉ đến của gói tin sẽ là địa chỉ quảng bá.

Có hai loại là: Local Broadcast và Global Broadcast

2.6.2. Sử dụng Broadcasting để gửi dữ liệu đến nhiều máy trong mạng cục bộ

Gửi gói dữ liệu Broadcast

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadBroadcast {
    public static void Main() {
        Socket sock = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
        IPEndPoint iep = new IPEndPoint(IPAddress.Broadcast, 9050);
        byte[] data = Encoding.ASCII.GetBytes("This is a test message");
        sock.SendTo(data, iep);
        sock.Close();
   }
```

```
}
```

Chúng ta phải thiết lập như sau:

```
class Broadcst {
      public static void Main() {
         Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
                        ProtocolType.Udp);
         IPEndPoint iep1 = new IPEndPoint(IPAddress.Broadcast, 9050);
         IPEndPoint iep2 = new IPEndPoint(IPAddress.Parse("192.168.1.255"), 9050);
         string hostname = Dns.GetHostName();
         byte[] data = Encoding.ASCII.GetBytes(hostname);
         sock.SetSocketOption(SocketOptionLevel.Socket,
                        SocketOptionName.Broadcast, 1);
         sock.SendTo(data, iep1);
         sock.SendTo(data, iep2);
         sock.Close();
      }
}
```

Nhận gói dữ liệu Broadcast

```
class RecvBroadcst {
   public static void Main()    {
      Socket sock = new Socket(AddressFamily.InterNetwork,SocketType.Dgram,
        ProtocolType.Udp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
      sock.Bind(iep);    EndPoint ep = (EndPoint)iep;
      Console.WriteLine("Ready to receive…");  byte[] data = new byte[1024];
      int recv = sock.ReceiveFrom(data, ref ep);
      string stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine("received: {0} from: {1}", stringData, ep.ToString());
      data = new byte[1024];  recv = sock.ReceiveFrom(data, ref ep);
      stringData = Encoding.ASCII.GetString(data, 0, recv);
      Console.WriteLine("received: {0} from: {1}",stringData, ep.ToString());
      sock.Close();
   }
}
```

2.6.3. Multicasting là gì?

Một địa chỉ multicast cho phép thiết bị gửi dữ liệu tới một tập xác định trước các host, được biết đến như các nhóm multicast, trong các mạng con khác nhau.

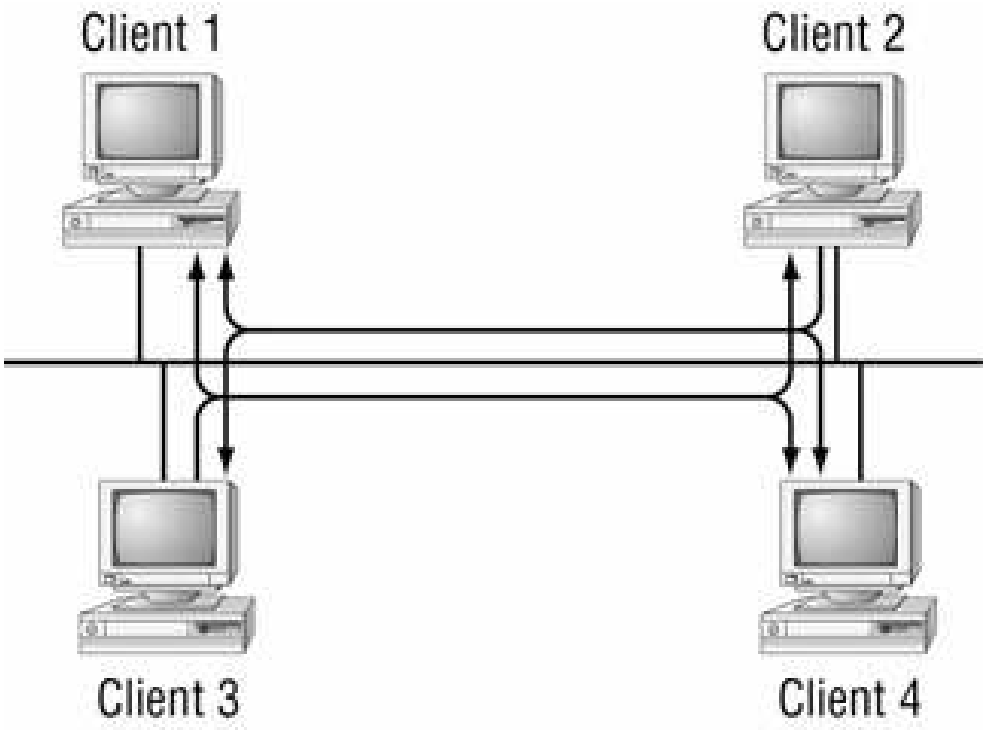
Một số địa chỉ Multicast

Địa chỉ multicast	**Chức năng**
224.0.0.0	Địa chỉ cơ sở
224.0.0.1	Tất cả các hệ thống trên mạng con này
224.0.0.2	Tất cả các Router trên mạng con này

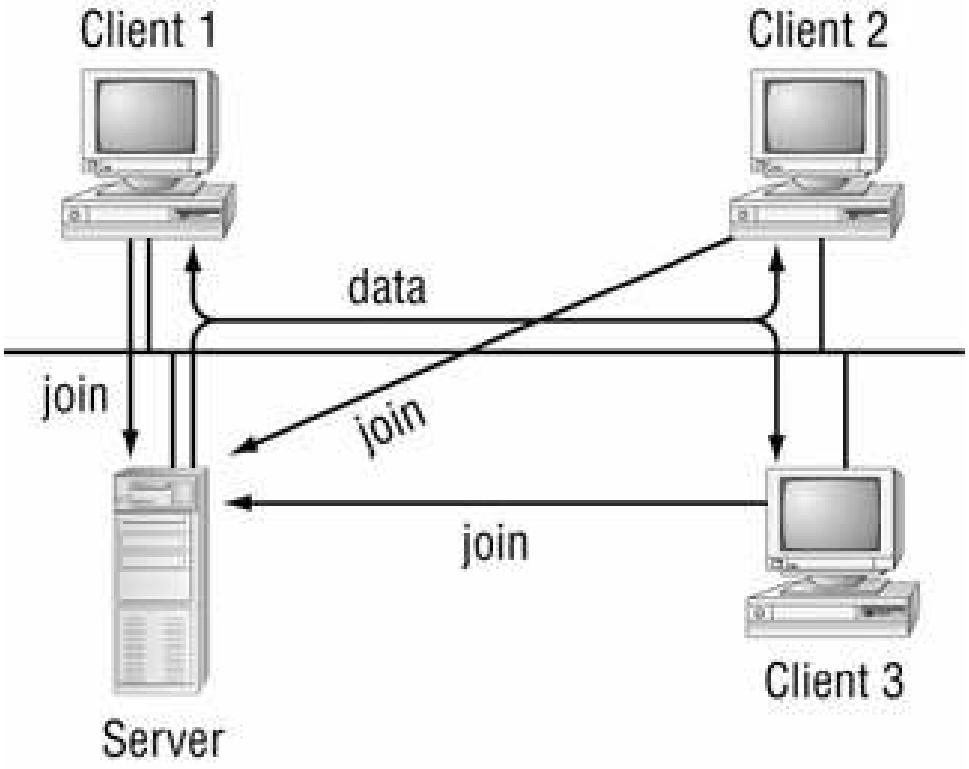
224.0.0.5	Các DR trong OSPF
224.0.1.9	Nhóm địa chỉ RIPv2
224.0.1.24	Nhóm địa chỉ WINS server

Có 2 kỹ thuật Multicast được sử dụng

+ Peer to Peer



+ Central Server



2.6.4. Socket Multicasting trong .Net

- Sử dụng phương thức SetSocketOption()

- Socket option có thể được sử dụng để
  - Thêm một Socket vào nhóm Multicast
  - Loại một Socket khỏi nhóm Multicast
  - SetSocketOption(SocketOptionLevel,SocketOptionName, optionValue)
  - SocketOptionName
    - AddMembership
    - DropMembership
- Sử dụng phương thức SetSocketOption()
- Socket option có thể được sử dụng để
  - optionValue là một đối tượng của lớp MulticastOption

  MulticastOption(IPAddress) MulticastOption(IPAddress,IPAddress)
- Ví dụ thêm một Socket vào nhóm Multicast 224.100.0.1

  sock.SetSocketOption(SocketOptionLevel.IP,

  SocketOptionName.AddMembership, new

  MulticastOption(IPAddress.Parse("224.100.0.1"));

Gửi dữ liệu Multicast

```
class MultiSend{
   public static void Main()   {
      Socket server = new Socket(AddressFamily.InterNetwork,
            SocketType.Dgram, ProtocolType.Udp);
      IPEndPoint iep = new              IPEndPoint(IPAddress.Parse("224.100.0.1"), 9050);
      byte[] data = Encoding.ASCII.GetBytes("This is a test message");
      server.SendTo(data, iep);
      server.Close();
   }
}
```

Nhận dữ liệu Multicast

```
class MultiRecv{
   public static void Main()  {
  Socket sock = new Socket(AddressFamily.InterNetwork,  SocketType.Dgram,
ProtocolType.Udp);
  Console.WriteLine("Ready to receive…");
  IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
  EndPoint ep = (EndPoint)iep;
  sock.Bind(iep);
  sock.SetSocketOption(SocketOptionLevel.IP,
        SocketOptionName.AddMembership,
                    new MulticastOption(IPAddress.Parse("224.100.0.1")));
  byte[] data = new byte[1024];
  int recv = sock.ReceiveFrom(data, ref ep);
  string stringData = Encoding.ASCII.GetString(data, 0, recv);
  Console.WriteLine("received: {0} from: {1}", stringData, ep.ToString());
  sock.Close();
  }
}
```

Gửi dữ liệu Multicast với TTL

```
class NewMultiSend{
   public static void Main()    {
      Socket server = new Socket(AddressFamily.InterNetwork,
                SocketType.Dgram, ProtocolType.Udp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9051);
      IPEndPoint iep2 = new IPEndPoint(IPAddress.Parse("224.100.0.1"), 9050);
      server.Bind(iep);
      byte[] data = Encoding.ASCII.GetBytes("This is a test message");
      server.SetSocketOption(SocketOptionLevel.IP,
        SocketOptionName.AddMembership,
              new MulticastOption(IPAddress.Parse("224.100.0.1")));
      server.SetSocketOption(SocketOptionLevel.IP,
      SocketOptionName.MulticastTimeToLive, 50);
      server.SendTo(data, iep2);
      server.Close();
   }
}
```

- Multicast với lớp UdpClient
  - JoinMulticastGroup()
  - DropMulticastGroup()
- JoinMulticastGroup() là phương thức overload
  - JoinMulticastGroup(IPAddress)
  - JoinMulticastGroup(IPAddress, int)

```
class UdpClientMultiSend{
   public static void Main()    {
      UdpClient sock = new UdpClient();
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("224.100.0.1"), 9050);
      byte[] data = Encoding.ASCII.GetBytes("This is a test message");
      sock.Send(data, data.Length, iep);
      sock.Close();
   }
}
class UdpClientMultiRecv
{
   public static void Main()
   {
      UdpClient sock = new UdpClient(9050);
      Console.WriteLine("Ready to receive…");
      sock.JoinMulticastGroup(IPAddress.Parse("224.100.0.1"), 50);
      IPEndPoint iep = new IPEndPoint(IPAddress.Any, 0);
      byte[] data = sock.Receive(ref iep);
      string stringData = Encoding.ASCII.GetString(data, 0, data.Length);
      Console.WriteLine("received: {0} from: {1}", stringData, iep.ToString());
      sock.Close();
   }
}
```

2.7 Bài tập áp dụng

```
class MulticastChat : Form{
   TextBox newText;
   ListBox results;
   Socket sock;
   Thread receiver;
   IPEndPoint multiep = new IPEndPoint(IPAddress.Parse("224.100.0.1"), 9050);
   public MulticastChat()    {
      Text = "Multicast Chat Program";
      Size = new Size(400, 380);
      Label label1 = new Label();
      label1.Parent = this;
      label1.Text = "Enter text string:";
      label1.AutoSize = true;
      label1.Location = new Point(10, 30);
      newText = new TextBox();
      newText.Parent = this;
      newText.Size = new Size(200, 2 * Font.Height);
      newText.Location = new Point(10, 55);
      results = new ListBox();
      results.Parent = this;
      results.Location = new Point(10, 85);
      results.Size = new Size(360, 18 * Font.Height);
      Button sendit = new Button();
      sendit.Parent = this;
      sendit.Text = "Send";
      sendit.Location = new Point(220, 52);
      sendit.Size = new Size(5 * Font.Height, 2 * Font.Height);
      sendit.Click += new EventHandler(ButtonSendOnClick);
Button closeit = new Button();
      closeit.Parent = this;
      closeit.Text = "Close";
      closeit.Location = new Point(290, 52);
      closeit.Size = new Size(5 * Font.Height, 2 * Font.Height);
      closeit.Click += new EventHandler(ButtonCloseOnClick);
      sock = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,
            ProtocolType.Udp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
      sock.Bind(iep);
      sock.SetSocketOption(SocketOptionLevel.IP,
        SocketOptionName.AddMembership,
```

```
                new MulticastOption(IPAddress.Parse("224.100.0.1")));
        receiver = new Thread(new ThreadStart(packetReceive));
        receiver.IsBackground = true;
        receiver.Start();
    }
    void ButtonSendOnClick(object obj, EventArgs ea)    {
        byte[] message = Encoding.ASCII.GetBytes(newText.Text);
        newText.Clear();
        sock.SendTo(message, SocketFlags.None, multiep);
    }
    void ButtonCloseOnClick(object obj, EventArgs ea)    {
        receiver.Abort();
        sock.Close();
        Close();
    }
void packetReceive()    {
        EndPoint ep = (EndPoint)multiep;
        byte[] data = new byte[1024];
        string stringData;
        int recv;
        while (true)        {
            recv = sock.ReceiveFrom(data, ref ep);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            results.Items.Add("from " + ep.ToString() + ": " + stringData);
        }
    }
    public static void Main()    {
        Application.Run(new MulticastChat());
    }
}
```

CHƯƠNG 3: XÂY DỰNG ỨNG DỤNG MẠNG

3.1. Giao thức ICMP

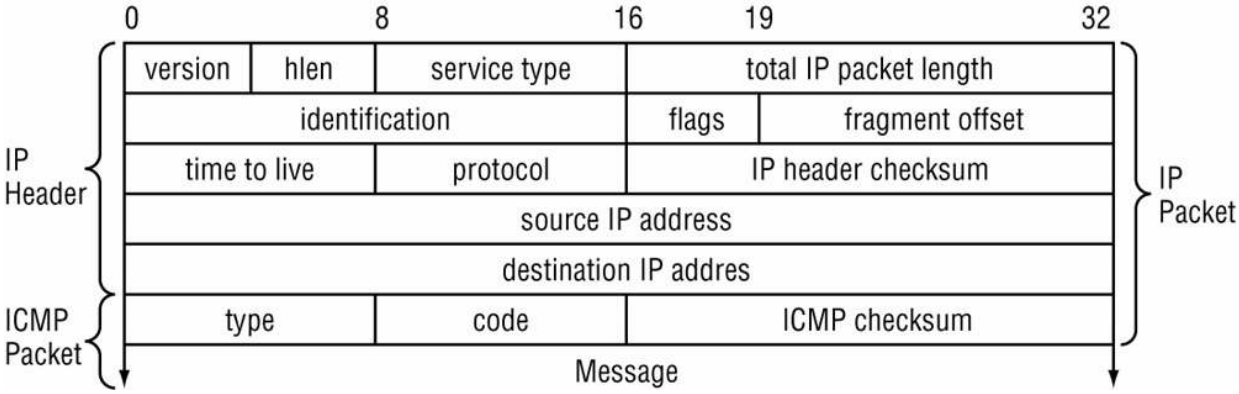
Giới thiệu giao thức ICMP (Internetwork Control Message Protocol)

- Giao thức ICMP hoạt động trên layer 2 - Internetwork trong mô hình TCP/IP hoặc layer 3 - Network trong mô hình OSI

Cho phép kiểm tra và xác định lỗi của Layer 3 Internetwork trong mô hình TCP/IP bằng cách định nghĩa ra các loại thông điệp có thể sử dụng để xác định xem mạng hiện tại có thể truyền được gói tin hay không.

Trong thực tế, ICMP cần các thành phần của mọi gói tin IP để có thể hoạt động được.

Cấu trúc của gói tin IP và ICMP



+ Type: có thể là một query hay một lỗi
+ Code: Xác định đây là loại query hay thông điệp lỗi
+ Checksum: Kiểm tra và sửa lỗi cho dữ liệu ICMP
+ Message: Tuỳ thuộc vào Type và Code

3.1.1. Sử dụng Raw Socket

Gói tin ICMP không sử dụng TCP hoặc UDP nên chúng ta không thể sử dụng các lớp được hỗ trợ như TcpClient hay UdpClient mà phải sử dụng một Raw Socket

Muốn tạo Raw Socket khi tạo ra Socket bạn sử dụng SocketType.Raw, giao thức ICMP

Tạo Raw Socket như sau

Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);

Raw Socket Format

Value	Description
Ggp	Gateway-to-Gateway Protocol
Icmp	Internet Control Message Protocol

Idp	IDP Protocol
Igmp	Internet Group Management Protocol
IP	A raw IP packet
Ipx	Novell IPX Protocol
ND	Net Disk Protocol
Pup	Xerox PARC Universal Protocol (PUP)
Raw	A raw IP packet
Spx	Novell SPX Protocol
SpxII	Novell SPX Version 2 Protocol
Unknown	An unknown protocol
Unspecified	An unspecified protocol

- Gửi gói dữ liệu Raw
  - ICMP là giao thức không hướng kết nối
  - Sử dụng phương thức SendTo() của lớp Socket để gửi
  - Cổng trong giao thức ICMP không quan trọng

```
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.2"), 0);
sock.SendTo(packet, iep);
```

- Nhận gói dữ liệu Raw
  - Sử dụng phương thức ReceiveForm cửa lớp Socket
  - Dữ liệu nhận về là một gói tin IP chúng ta phải tách ra để lấy gói tin ICMP

Raw Socket không tự động định dạng gói tin ICMP cho chúng ta. Chúng ta phải tự làm

Data Variable	**Size**	**Type**
Type	1 byte	Byte
Code	1 byte	Byte
Checksum	2 bytes	Unsigned 16-bit integer
Message	multibyte	Byte array

Định nghĩa lớp và phương thức khởi tạo mặc định

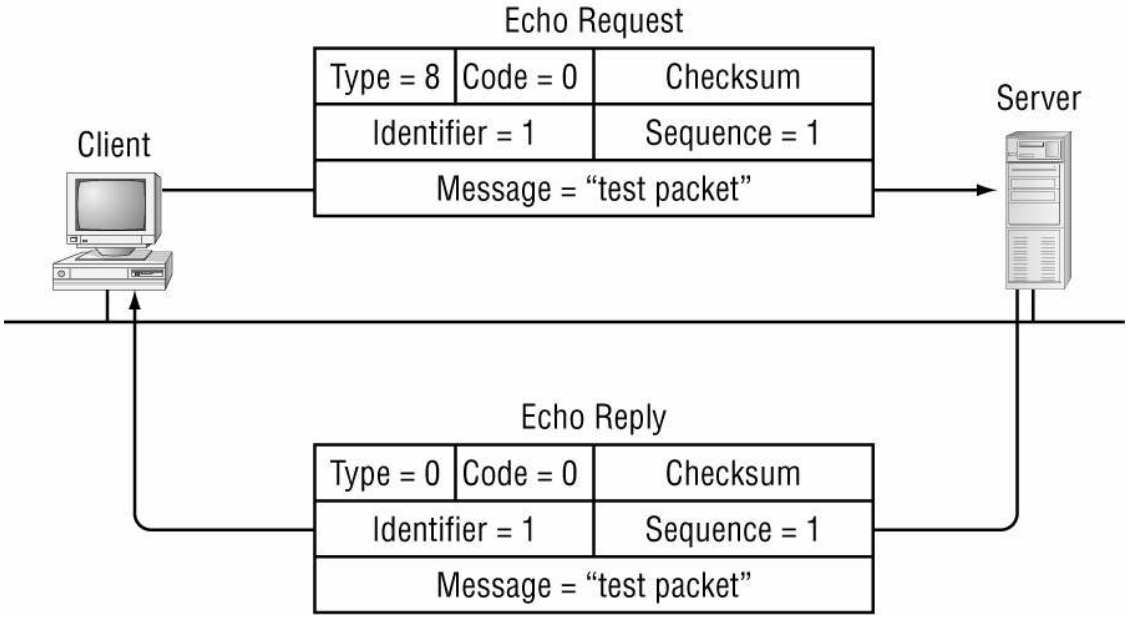
```
class ICMP {
   public byte Type;
   public byte Code;
   public UInt16 Checksum;
   public int Messagesize;
   public byte[] Message = new byte[1024];
   public ICMP() {
   }
}
```

- Tạo ra gói tin ICMP

```
ICMP packet = new ICMP();
```

```
            packet.Type = 0x08;
            packet.Code = 0x00;
            packet.Checksum = 0;
public ICMP(byte[] data, int size) {
        Type = data[20];
        Code = data[21];
        Checksum = BitConverter.ToUInt16(data, 22);
        MessageSize = size - 24;
        Buffer.BlockCopy(data, 24, Message, 0, MessageSize);
    }
public byte[] getBytes() {
        byte[] data = new byte[MessageSize + 9];
        Buffer.BlockCopy(BitConverter.GetBytes(Type), 0, data, 0, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Code), 0, data, 1, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Checksum), 0, data, 2, 2);
        Buffer.BlockCopy(Message, 0, data, 4, MessageSize);
        return data;
    }
public UInt16 getChecksum() {
        UInt32 chcksm = 0;
        byte[] data = getBytes();
        int packetsize = MessageSize + 8;
        int index = 0;
        while (index < packetsize) {
            chcksm += Convert.ToUInt32(BitConverter.ToUInt16(data, index));
            index += 2;
        }
        chcksm = (chcksm >> 16) + (chcksm & 0xffff);
        chcksm += (chcksm >> 16);
        return (UInt16)(~chcksm);
    }
```

3.1.2. Sử dụng giao thức ICMP và Raw Socket để xây dựng chương trình Ping




```
class Program {
   static void Main(string[] args) {
      byte[] data = new byte[1024];
      int recv;
      Socket host = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
         ProtocolType.Icmp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse(argv[0]), 0);
      EndPoint ep = (EndPoint)iep;
      ICMP packet = new ICMP();
      packet.Type = 0x08;
      packet.Code = 0x00;
      packet.Checksum = 0;
      Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 0, 2);
      Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 2, 2);
      data = Encoding.ASCII.GetBytes("test packet");
      Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);

packet.MessageSize = data.Length + 4;
      int packetsize = packet.MessageSize + 4;
      UInt16 chcksum = packet.getChecksum();
      packet.Checksum = chcksum;
      host.SetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout, 3000);
      host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
      try {
         data = new byte[1024];
         recv = host.ReceiveFrom(data, ref ep);
      } catch (SocketException) {
         Console.WriteLine("No response from remote host");
         return;
      }
ICMP response = new ICMP(data, recv);
      Console.WriteLine("response from: {0}", ep.ToString());
      Console.WriteLine(" Type {0}", response.Type);
      Console.WriteLine(" Code: {0}", response.Code);
      int Identifier = BitConverter.ToInt16(response.Message, 0);
```

```
      int Sequence = BitConverter.ToInt16(response.Message, 2);
      Console.WriteLine(" Identifier: {0}", Identifier);
      Console.WriteLine(" Sequence: {0}", Sequence);
      string stringData = Encoding.ASCII.GetString(response.Message, 4,
         response.MessageSize - 4);
      Console.WriteLine(" data: {0}", stringData);
      host.Close();
   }
}
```

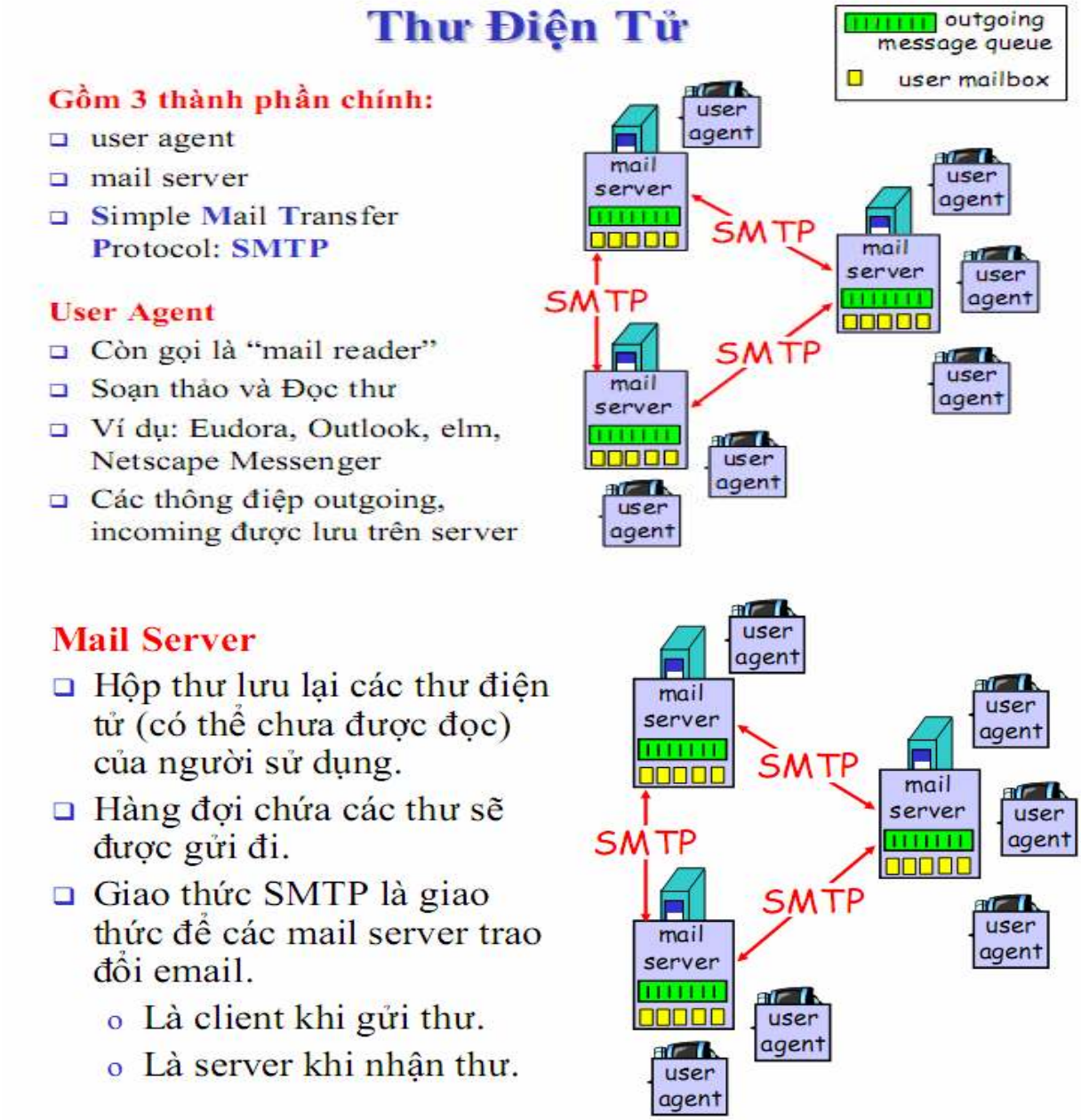
3.1.3. Sử dụng giao thức ICMP và Raw Socket để xây dựng chương trình TraceRoute

```
class TraceRoute {
   public static void Main(string[] argv) {
      byte[] data = new byte[1024];
      int recv, timestart, timestop;
      Socket host = new Socket(AddressFamily.InterNetwork,
           SocketType.Raw, ProtocolType.Icmp);
      IPHostEntry iphe = Dns.Resolve(argv[0]);
      IPEndPoint iep = new IPEndPoint(iphe.AddressList[0], 0);
      EndPoint ep = (EndPoint)iep;
      ICMP packet = new ICMP();
      packet.Type = 0x08;
      packet.Code = 0x00;
      packet.Checksum = 0;
      Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 0, 2);
      Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 2, 2);
      data = Encoding.ASCII.GetBytes("test packet");
Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);
      packet.MessageSize = data.Length + 4;
      int packetsize = packet.MessageSize + 4;
      UInt16 chcksum = packet.getCchecksum();
      packet.Checksum = chcksum;
      host.SetSocketOption(SocketOptionLevel.Socket,
              SocketOptionName.ReceiveTimeout, 3000);
      int badcount = 0;
      for (int i = 1; i < 50; i++) {
         host.SetSocketOption(SocketOptionLevel.IP,
              SocketOptionName.IpTimeToLive, i);
         timestart = Environment.TickCount;
         host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
try {
            data = new byte[1024];
            recv = host.ReceiveFrom(data, ref ep);
            timestop = Environment.TickCount;
            ICMP response = new ICMP(data, recv);
            if (response.Type == 11)
               Console.WriteLine("hop {0}: response from {1}, {2}ms",
                 i, ep.ToString(), timestop - timestart);
            if (response.Type == 0) {
               Console.WriteLine("{0} reached in {1} hops, {2}ms.",
                 ep.ToString(), i, timestop - timestart);
               break;
```

```
            }
badcount = 0;
        } catch (SocketException) {
            Console.WriteLine("hop {0}: No response from remote host", i);
            badcount++;
            if (badcount == 5) {
                Console.WriteLine("Unable to contact remote host");
                break;
            }
        }
      }
      host.Close();
   }
}
```

3.2. Giao thức SMTP, POP3

3.2.1. Cơ bản về hệ thống Mail và giao thức SMTP, POP3



*** Giao thức SMTP**

Một số lệnh cơ bản của giao thức SMTP:

Lệnh	Mô tả
HELO	Hello. Sử dụng để xác định người gửi điện. Lệnh này này đi kèm với tên của host gửi điện. Trong ESTMP (extended protocol), thì lệnh này sẽ là EHLO.
MAIL	Khởi tạo một giao dịch gửi thư. Nó kết hợp "from" để xác định người gửi thư.
RCPT	Xác định người nhận thư.
DATA	Thông báo bắt đầu nội dung thực sự của bức điện (phần thân của thư). Dữ liệu được mã thành dạng mã 128-bit ASCII và nó được kết thúc với một dòng đơn chứa dấu chấm (.).
RSET	Huỷ bỏ giao dịch thư
VRFY	Sử dụng để xác thực người nhận thư.
NOOP	Nó là lệnh "no operation" xác định không thực hiện hành động gì
QUIT	Thoát khỏi tiến trình để kết thúc
SEND	Cho host nhận biết rằng thư còn phải gửi đến đầu cuối khác.

Một số lệnh không yêu cầu phải có được xác định bằng RFC 821

SOML	Send or mail. Báo với host nhận thư rằng thư phải gửi đến đấu cuối khác hoặc hộp thư.
SAML	Send and mail. Nói với host nhận rằng bức điện phải gửi tới người dùng đầu cuối và hộp thư.
EXPN	Sử dụng mở rộng cho một mailing list.
HELP	Yêu cầu thông tin giúp đỡ từ đầu nhận thư.
TURN	Yêu cầu để host nhận giữ vai trò là host gửi thư.

- Mã trạng thái của SMTP

Khi một MTA gửi một lệnh SMTP tới MTA nhận thì MTA nhận sẽ trả lời với một mã trạng thái để cho người gửi biết đang có việc gì xẩy ra đầu nhận. Và dưới đây là bảng mã trạng thái của SMTP theo tiêu chuẩn RFC 821. Mức độ của trạng thái được

xác định bởi số đầu tiên của mã (5xx là lỗi nặng, 4xx là lỗi tạm thời, 1xx–3xx là hoạt động bình thường).

- Một số mã trạng thái của SMTP

211 Tình trạng hệ thống, hay reply giúp đỡ hệ thống

214 Thông điệp giúp đỡ.

220 <domain> dịch vụ sẵn sàng

221 <domain> dịch vụ đóng kênh giao chuyển

250 Hành động mail yêu cầu OK, hoàn thành

251 User không cục bộ; sẽ hướng đến <forward-path>

354 Khởi động việc nhập mail; kết thúc với <CLRF>. <CLRF>

421 <domain> dịch vụ không sử dụng được, đóng kênh giao chuyển

450 Không lấy hành động mail yêu cầu; mailbox không hiệu lực

451 Không nhận hành động được yêu cầu; lưu trữ của hệ thống không đủ.

500 Lỗi cú pháp; không chấp nhận lệnh

501 Lỗi cú pháp trong tham số hay đối số

502 Lệnh không được cung cấp

503 Dòng lệnh sai

504 Tham số của dòng lệnh không được cung cấp

550 Không nhận hành động được yêu cầu ; mailbox không hiệu lực

[như mailbox không tìm thấy hay không truy cập được]

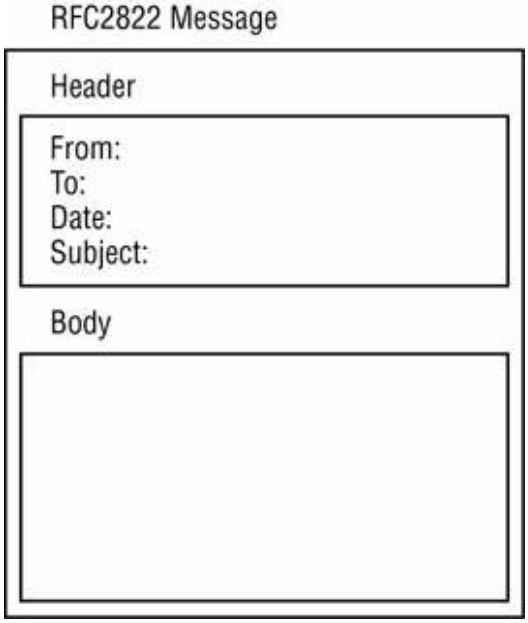
551 User không cục bộ; vui lòng thử <forward-path>

552 Bỏ qua hành động mà mail yêu cầu, vượt quá chỉ định lưu trữ

554 Không nhận hành động được yêu cầu; tên mailbox không được

chấp nhận. [như sai cú pháp mailbox] giao chuyển sai.

- Định dạng của một bức thư thông thường không có phần đính kèm như sau:



* Giao thức POP3

Giao thức dùng để lấy thư, POP3 Server lắng nghe trên cổng 110, mô tả trong RFC 1939

- Một số lệnh của POP3

USER Xác định username

PASS Xác định password

STAT Yêu cầu về trạng thái của hộp thư như số lượng thư và độ lớn của thư

LIST Hiện danh sách của thư

RETR Nhận thư

DELE Xoá một bức thư xác định

NOOP Không làm gì cả

RSET Khôi phục lại như thư đã xoá (rollback)

QUIT Thực hiện việc thay đổi và thoát ra

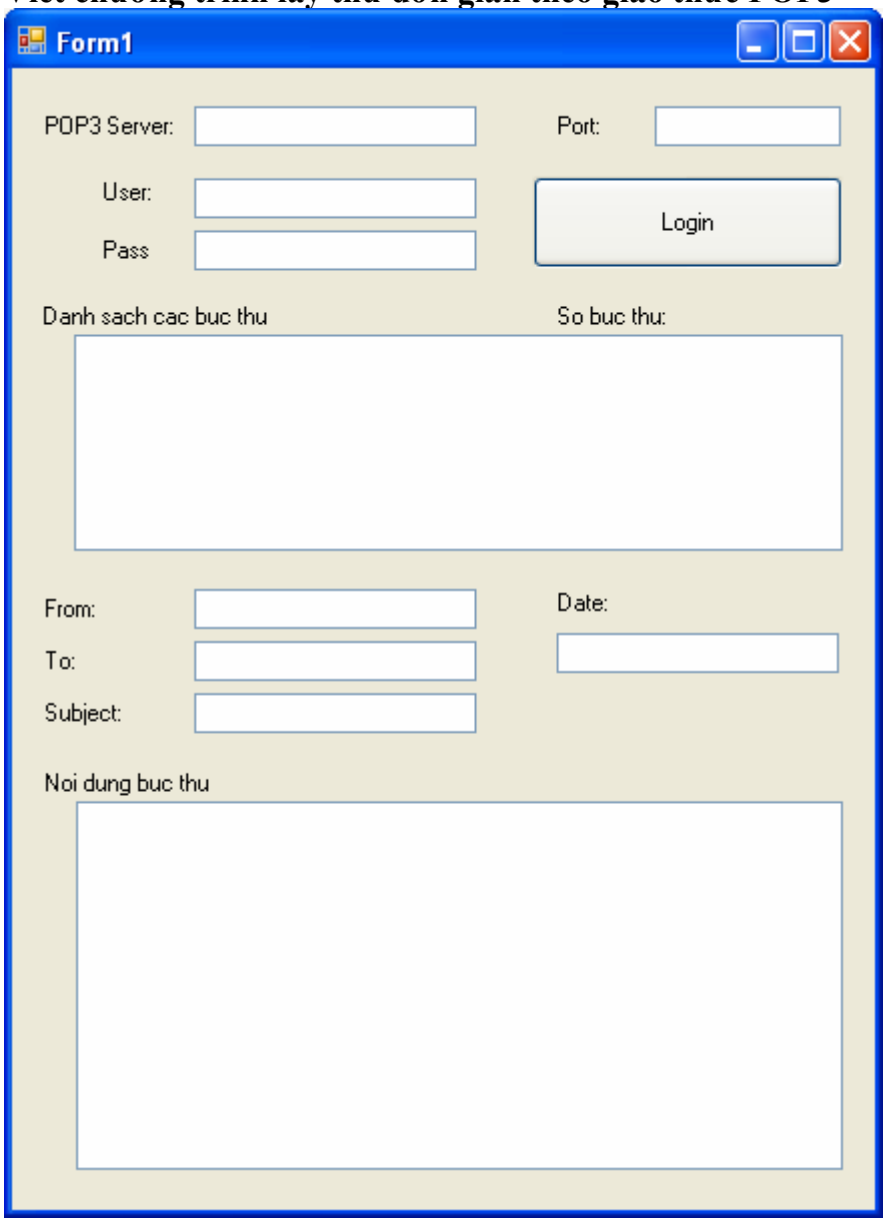
3.2.2. Cài đặt SMTP, POP3 Client/Server

Viết chương trình gửi Mail đơn giản theo giao thức SMTP

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
class Program {
   static void Main(string[] args) {
      string nguoigui, nguoinhan, tieude,body;
      string diachimaychu;
      int portmaychu;
      Console.Write("Nhap di chu SMTP Server:");
      diachimaychu = Console.ReadLine();
      Console.Write("Nhap cong cua may chu:");
      portmaychu = int.Parse(Console.ReadLine());
      Console.Write("Nhap dia chi nguoi gui:");
      nguoigui = Console.ReadLine();
      Console.Write("Nhap dia chi nguoi nhan:");
      nguoinhan = Console.ReadLine();
      Console.Write("Nhap tieu de buc thu:");
      tieude = Console.ReadLine();
      Console.Write("Nhap noi dung buc thu:");
      body= Console.ReadLine();
      //Tao Endpoit cua may chu
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse(diachimaychu), portmaychu);
      TcpClient client = new TcpClient();
      client.Connect(iep);
      string Data = "Helo";
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      sw.WriteLine(Data);
      sw.Flush();
      //Doc thong bao tu Server gui ve va xu ly neu can thiet
      Console.WriteLine(sr.ReadLine());
```

```
        //Gui dia chi nguyoi gui
        Data = "MAIL FROM: <" + nguoigui + ">";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Gui dia chi nguyoi gui
        Data = "RCPT TO: <" + nguoinhan + ">";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Bat dau gui noi dung buc thu
        Data = "Data";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Gui noi dung buc thu
        Data = "SUBJECT:" + tieude + "\r\n" + body + "\r\n" + "." + "\r\n";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Ngat ket noi
        Data = "QUIT";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        sr.Close();
        sw.Close();
        client.Close();
        Console.ReadLine();
    }
}
```

Viết chương trình lấy thư đơn giản theo giao thức POP3




```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace MyPop3 {
   public partial class Form1 : Form {
      TcpClient popclient;
      StreamReader sr;
      StreamWriter sw;
      public Form1() {
         InitializeComponent();
         CheckForIllegalCrossThreadCalls = false;
```

```
    }

    private void btLogin_Click(object sender, EventArgs e) {
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse(txtPOP.Text),
int.Parse(txtPort.Text));
        popclient = new TcpClient();
        popclient.Connect(iep);
        sr = new StreamReader(popclient.GetStream());
        sw = new StreamWriter(popclient.GetStream());
        sr.ReadLine();
        string data = "";
        data = "User " + txtUser.Text;
        sw.WriteLine(data);
        sw.Flush();
        sr.ReadLine();
        data = "PASS " + txtPass.Text;
        sw.WriteLine(data);
        sw.Flush();
        sr.ReadLine();
        data = "LIST";
        sw.WriteLine(data);
        sw.Flush();
        lstHeader.Items.Clear();
        string s = sr.ReadLine();
        char[] ch = { ' ' };
        string[] tam = s.Split(ch);
        //MessageBox.Show("so buc thu la:" + tam[1]);
        while ((s = sr.ReadLine()) != ".") {
            lstHeader.Items.Add(s);
        }
    }

    private void lstHeader_SelectedIndexChanged(object sender, EventArgs e) {
        int i = lstHeader.SelectedIndex + 1;
        //Lay buc thu ve va tien hanh phan tich
        string data = "RETR " + i.ToString();
        sw.WriteLine(data);
        sw.Flush();
        string s;
        //MessageBox.Show(sr.ReadLine());
        //Lay phan header
        while ((s = sr.ReadLine().Trim()) != null) {
            //MessageBox.Show(s);
            if (s.Length == 0) break;
            if (s.ToUpper().StartsWith("DATE")) {
                DateTime dt=DateTime.Parse(s.Substring(5, s.Length - 5));
                txtDate.Text = dt.ToShortDateString() +" " +dt.ToLongTimeString();
            }
            if (s.ToUpper().StartsWith("FROM"))
```

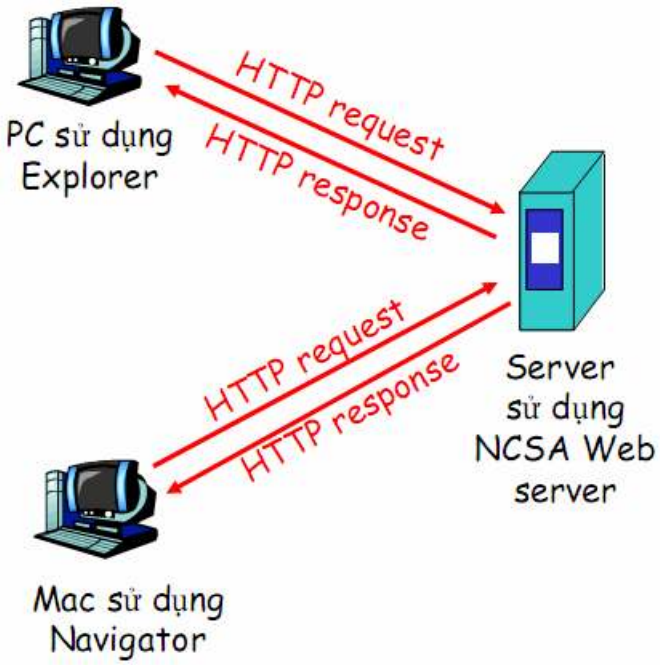
```
                txtFrom.Text = s.Substring(5, s.Length - 5);
            if (s.ToUpper().StartsWith("TO"))
                txtTo.Text = s.Substring(3, s.Length - 3);
            if (s.ToUpper().StartsWith("SUBJECT"))
                txtSubject.Text = s.Substring(8, s.Length - 8);
        }
        //Lay phan body
        textBox4.Clear();
        //MessageBox.Show("Lay body");
        while (!sr.EndOfStream) {
            s = sr.ReadLine().Trim();
            if (s.Equals(".")) break;
            textBox4.Text += s + "\r\n";
        }
        //MessageBox.Show("Het noi dung buc thu");
    }
  }
}
```

3.3. Giao thức HTTP

3.3.1. Cơ bản về giao thức HTTP

HTTP (Hypertext Transfer Protocol) giao thức truyền siêu văn bản. HTTP là giao thức tầng ứng dụng cho Web. Nó hoạt động theo mô hình client/server.

- Client: browser yêu cầu, nhận, hiển thị các đối tượng Web.
- Server: Web server gửi các đối tượng



Hai phiên bản của giao thức HTTP hiện được phổ biến là HTTP1.0 được đặc tả trong RFC 1945 và HTTP1.1 được đặc tả trong RFC 2068.

HTTP là giao thức "không trạng thái" server không lưu lại các yêu cầu của client.

HTTP sử dụng giao thức TCP của tầng giao vận. Các bước tiến hành từ khi client kết nối tới server sau đó gửi và nhận kết quả từ server gửi về như sau:

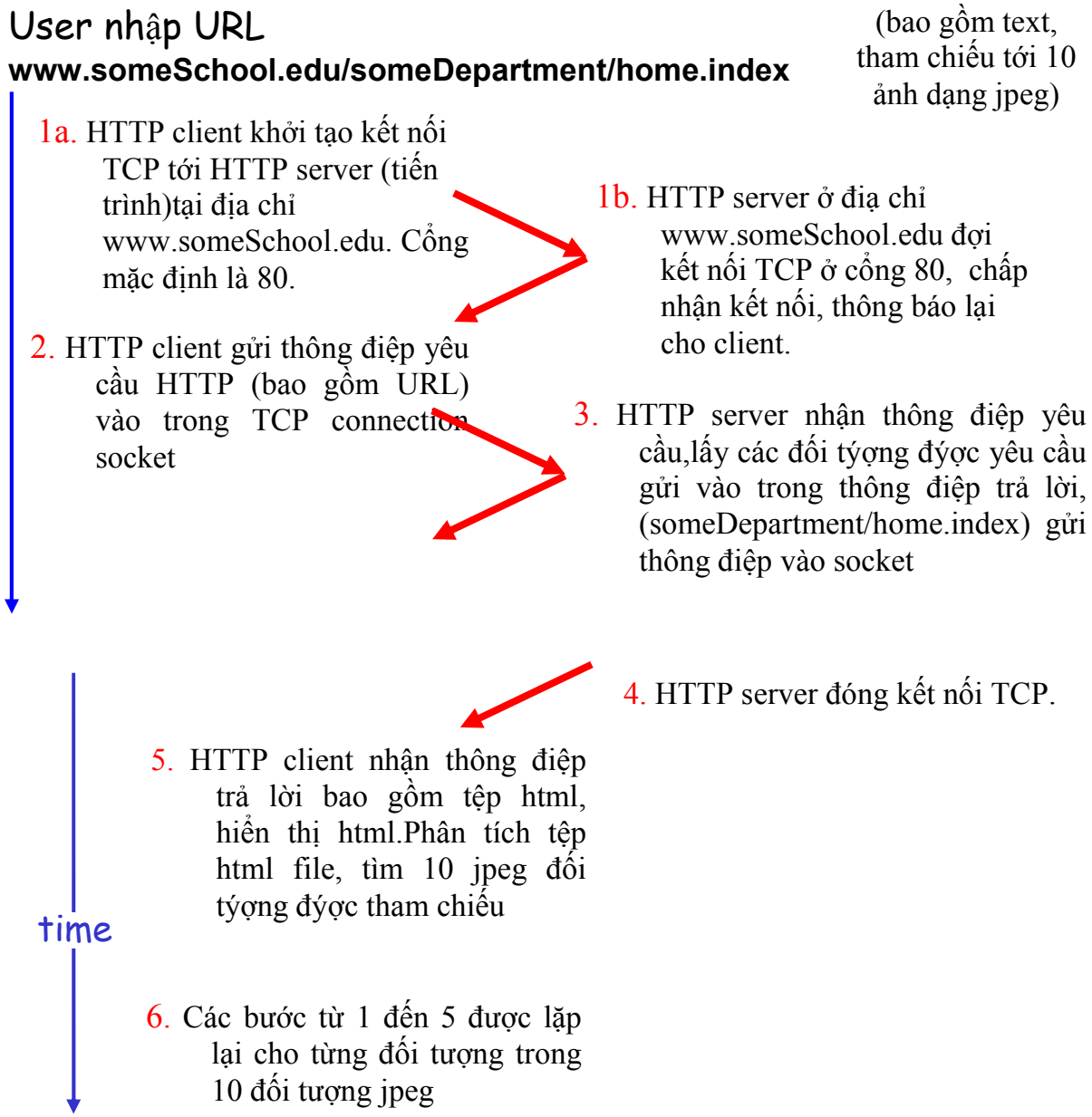
+ client khởi tạo kết nối TCP (tạo socket) với server, qua cổng 80

+ server chấp nhận kết nối TCP từ client

+ Các thông điệp HTTP (thông điệp tầng ứng dụng) được trao đổi giữa browser (HTTP client) và Web server (HTTP server)

+ Đóng kết nối TCP.

Chúng ta xem một ví dụ về quá trình trao đổi giữa browser và Web server như sau:



Có hai kiểu thông điệp HTTP là yêu cầu (Request) và trả lời (Response). Các thông điệp được định dạng kiểu mã ASCII.

Định dạng thông điệp yêu cầu HTTP

Dòng yêu cầu (lệnh GET, POST, HEAD)

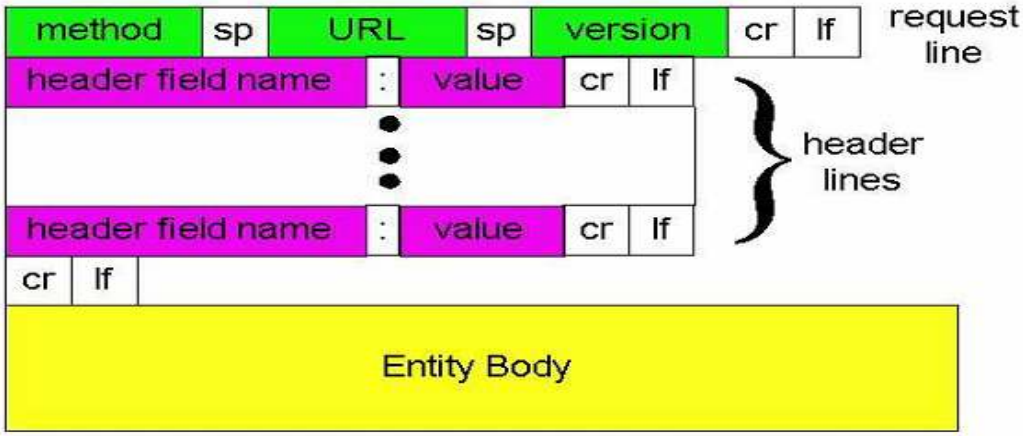
Những dòng header

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(CR,LF)

CR,LF,kí hiệu kết thúc thông điệp

Định dạng Thông điệp Yêu cầu HTTP



Định dạng thông điệp trả lời HTTP

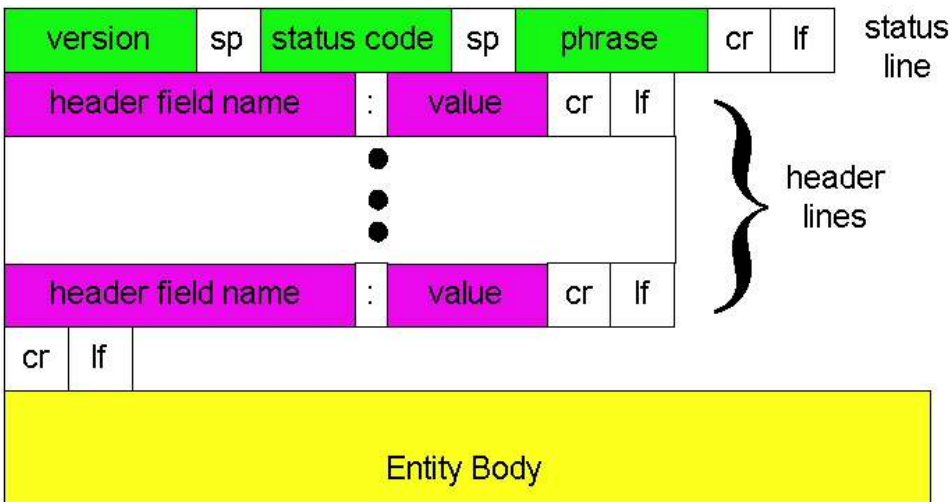
Dòng trạng thái (mã trạng thái)

Những dòng header

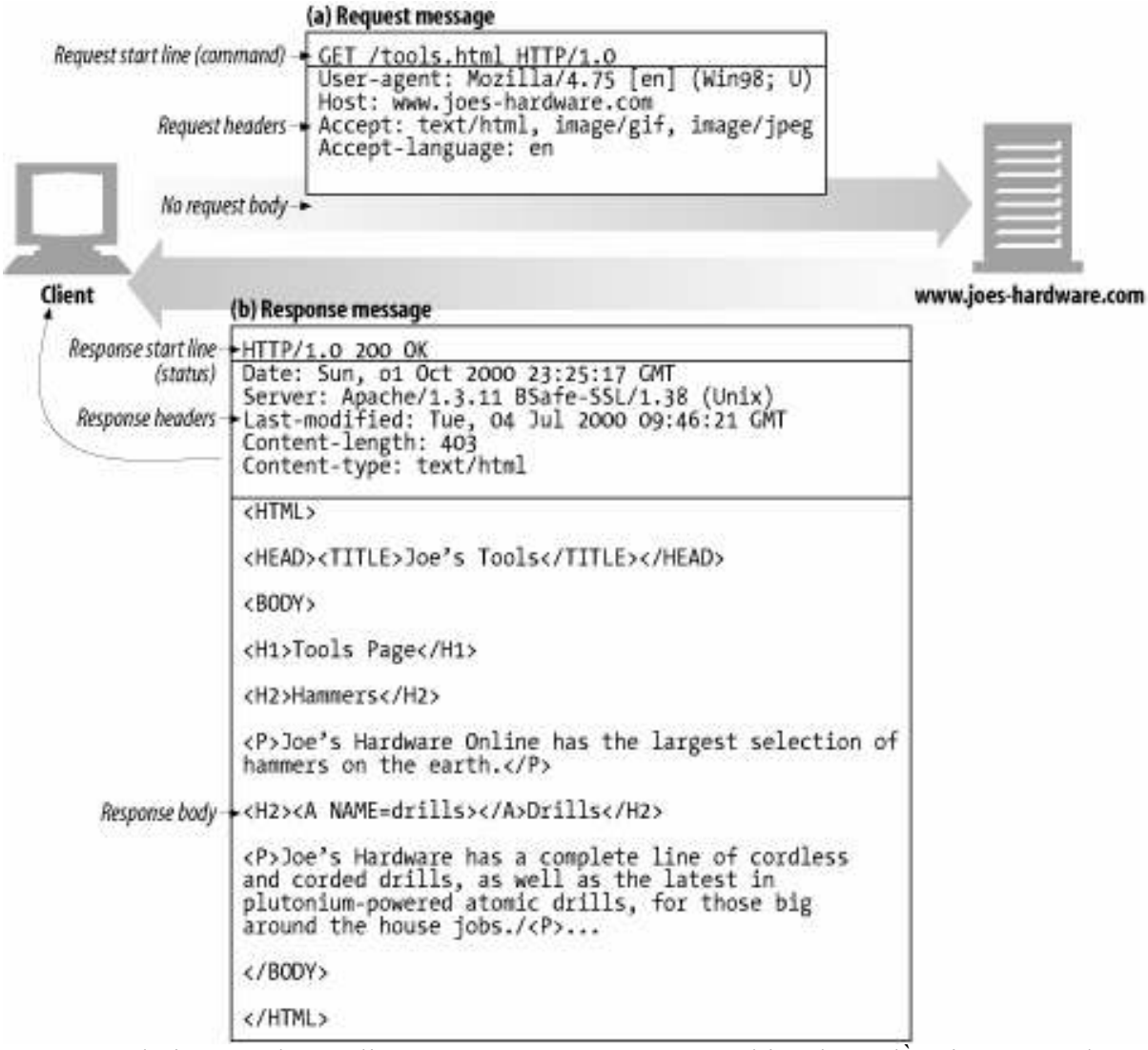
```
HTTP/1.0 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

dữ liệu,e.g., tệp html đýợc yêu cầu



Quá trình trao đổi giữa Browser và Web Server có thể được minh họa như hình sau:



Mã trạng thái trong thông điệp HTTP Response: Được ghi ở dòng đầu tiên trong thông điệp response từ server về client.

Một số mã thường gặp:

200 OK

Yêu cầu thành công,các đối tượng được yêu cầu ở phần sau thông điệp.

301 Moved Permanently

Đối tượng được yêu cầu đã được chuyển và địa chỉ mới của đối tượng được đặt trong trường Location:

400 Bad Request

Server không hiểu được thông điệp yêu cầu

404 Not Found

Tài liệu được yêu cầu không có trong server

505 HTTP Version Not Supported

Server không hỗ trợ version của giao thức HTTP.

- Để kiểm tra các lệnh của HTTP bên phía Client chúng ta có thể thực hiện như sau:

+ Telnet tới Web server

`telnet www.eurecom.fr 80` Tạo kết nối TCP ở cổng 80 (cổng mặc định cho HTTP server) at www.eurecom.fr.

+ Lệnh GET trong thông điệp HTTP

`GET /~ross/index.html HTTP/1.0` Gửi thông điệp yêu cầu lấy tệp Index.html trong thư mục ~ross về client

+ Xem thông điệp response được gửi về từ Server

- Gõ các lệnh khác để kiểm tra.

3.3.2. Cài đặt HTTP Client/Server

a. Chương trình HTTP Client

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Windows.Forms;

namespace HttpView {
   public partial class MainForm : Form
   {
      [STAThread]
      public static void Main(string[] args){
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new MainForm());
      }
      public MainForm(){
         //
         // The InitializeComponent() call is required for Windows Forms designer
support.
         //
         InitializeComponent();

         //
         // TODO: Add constructor code after the InitializeComponent() call.
         //
      }
      // Gửi yêu cầu và nhận về phản hồi từ web server
      void BtGoClick(object sender, EventArgs e){
         //Thêm tiền tố http:// nếu không có
         if (txtAddress.Text.StartsWith("http://", StringComparison.OrdinalIgnoreCase)
== false)
            txtAddress.Text = "http://" + txtAddress.Text;
         TcpClient client = new TcpClient();
         try{
            client.Connect(GetHost(txtAddress.Text), GetPort(txtAddress.Text));
         }
         catch{
```

```
            rtfBody.Text = "Không thể kết nối đến server";
            return;
        }
        StreamWriter OutStream = new StreamWriter(client.GetStream());
        StreamReader InpStream = new StreamReader(client.GetStream());
        // Gửi đi yêu cầu bằng phương thức GET
        OutStream.WriteLine("GET " + txtAddress.Text + " HTTP/1.0");
        OutStream.WriteLine("Content-Type:text/html");
        OutStream.WriteLine("Content-Language:en");
        OutStream.WriteLine();
        OutStream.Flush();
        //Lấy phần Header
        rtfHeader.Text = "";
        string s;
        while (null != (s = InpStream.ReadLine())){
            if (s.Equals("")) break;
            rtfHeader.Text += s;
        }
        //Lấy phần Body
        rtfBody.Text = "";
        int c;
        while (true){
            c = InpStream.Read();
            if (c == -1) break;
            rtfBody.Text += (char)c;
        }
        client.Close();
    }
    // Khi gõ Enter thì gửi đi yêu cầu
    void TxtAddressKeyPress(object sender, KeyPressEventArgs e){
        if ((int)e.KeyChar == 13)
            btGo.PerformClick();
    }
    // Lấy về tên máy trong URL
    internal string GetHost(string url){
        url = url.ToLower();
        Uri tmp = new Uri(url);
        return tmp.Host;
    }
    // Lấy về số hiệu cổng trong URL
    internal int GetPort(string url){
        Uri tmp = new Uri(url);
        return tmp.Port;
    }
  }
}
```

b. Chương trình HTTP Server

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

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

public class SimpleWebServer {
	public void StartListening(int port) {
		IPEndPoint LocalEndPoint = new IPEndPoint(IPAddress.Any, 8080);
		TcpListener server = new TcpListener(LocalEndPoint);
		server.Start();
		while (true) {
			TcpClient client = server.AcceptTcpClient(); // Cho doi ket noi
			processClientRequest(client);
		}
	}
	//Xử lý yêu cầu của mỗi máy khách
	private void processClientRequest(TcpClient client) {
		Console.WriteLine("May khach su dung cong: " +

(client.Client.RemoteEndPoint as IPEndPoint).Port + "; Dia chi IP may khach: " +

client.Client.RemoteEndPoint.ToString() + "\n");
		Console.WriteLine("Yeu cau: ");
		readRequest(client);
		sendResponse(client);
		client.Close();
	}
	//Đọc phần header của gói dữ liệu gửi từ máy khách
	private void readRequest(TcpClient client) {
		StreamReader request = new StreamReader(client.GetStream());
		String nextLine;

		while (null != (nextLine = request.ReadLine())) {
			if (nextLine.Equals("")) {
				break;
			} else {
				Console.WriteLine(nextLine);
			}
		}
		Console.WriteLine("----------------------------------------");
	}

	//Phản hồi yêu cầu cho máy khách
	private void sendResponse(TcpClient client) {
		StreamWriter response = new StreamWriter(client.GetStream());
		response.Write(_mainPage);
		response.Flush();
	}
	//Mội phản hồi về trình duyệt
	private const String _mainPage =
		"HTTP/1.0 200 OK\n" +
```

```
                "Content-type: text/html\n" +
                "Connection: close\n" +
                "\n<HTML>\n" +
                "<HEAD>\n" +
                "<TITLE>Hello World</TITLE>\n" +
                "</HEAD>\n" +
                "<BODY> <a href = \"localhost\"> Hello World </a></BODY>\n" +
                "</HTML> \n\n";
}

class TestWebServer {
        public static void Main(String[] args) {
                SimpleWebServer webServer1 = new SimpleWebServer();
                webServer1.StartListening(8080);
        }
}
```

3.4. Giao thức FTP

3.4.1. Cơ bản về giao thức FTP

Giao thức FTP là một giao thức trao đổi file khá phổ biến

Khái niệm và hoạt động cơ bản của FTP:

Những người phát triển giao thức FTP cần phải cân bằng nhu cầu giữa việc phát triển một bộ giao thức vừa có nhiều chức năng và vừa đơn giản trong triển khai. Do đó, FTP không đơn giản như "đứa em trai" của nó – là giao thức TFTP. Hoạt động của giao thức này có thể chia ra thành nhiều thành phần nhỏ, hoạt động cùng nhau để thực hiện các công việc như khởi tạo kết nối, truyền thông tin điều khiển và truyền lệnh.

Nội dung mà bài viết này hướng tới là đưa tới người đọc những khái niệm quan trọng nhất đằng sau giao thức FTP, cũng như giải thích tổng quan về nguyên lý hoạt động của nó.

1 - Mô hình hoạt động của FTP, các thành phần trong giao thức, và các thuật ngữ cơ bản

Giao thức FTP được mô tả một cách đơn giản thông qua mô hình hoạt động của FTP. Mô hình này chỉ ra các nguyên tắc mà một thiết bị phải tuân theo khi tham gia vào quá trình trao đổi file, cũng như về hai kênh thông tin cần phải thiết lập giữa các thiết bị đó. Nó cũng mô tả các thành phần của FTP được dùng để quản lý các kênh này ở cả hai phía – truyền và nhận. Do đó, mô hình này tạo cho ta một khởi điểm lý tưởng để xem xét hoạt động của FTP ở mức khái quát.

Tiến trình Server-FTP và User-FTP

FTP là một giao thức dạng client/server truyền thống, tuy nhiên thuật ngữ client thông thường được thay thế bằng thuật ngữ user – người dùng – do thực tế là người sử dụng mới là đối tượng trực tiếp thao tác các lệnh FTP trên máy clients. Bộ phần mềm FTP được cài đặt trên một thiết bị được gọi là một tiến trình. Phần mềm FTP được cài đặt trên máy Server được gọi là tiến trình Server-FTP, và phần trên máy client được gọi là tiến trình User-FTP.

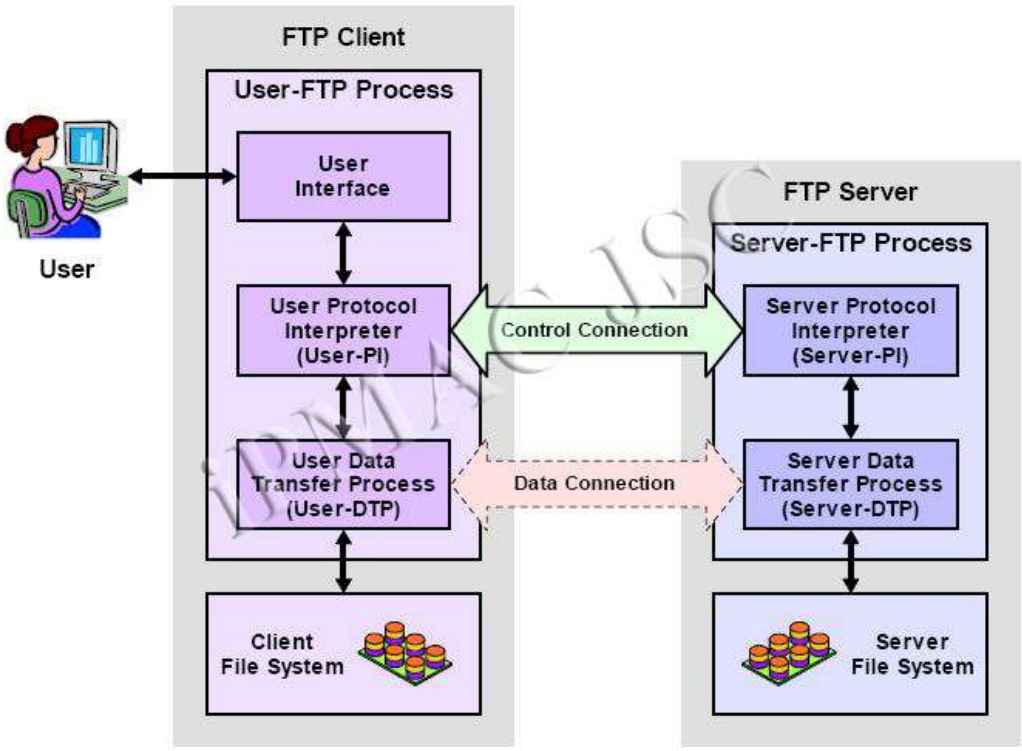
Kênh điều khiển và kênh dữ liệu trong FTP

Một khái niệm cốt yếu mà ta cần phải nắm về FTP là: mặc dù giao thức này sử dụng kết nối TCP, nhưng nó không chỉ dùng một kênh TCP như phần lớn các giao thức truyền thông khác. Mô hình FTP chia quá trình truyền thông giữa bộ phận Server với bộ phận client ra làm hai kênh logic:
- Kênh điều khiển: đây là kênh logic TCP được dùng để khởi tạo một phiên kết nối FTP. Nó được duy trì xuyên suốt phiên kết nối FTP và được sử dụng chỉ để truyền các thông tin điều khiển, như các lệnh và các hồi đáp trong FTP. Nó không được dùng để truyền file
- Kênh dữ liệu: Mỗi khi dữ liệu được truyền từ server tới client, một kênh kết nối TCP nhất định lại được khởi tạo giữa chúng. Dữ liệu được truyền đi qua kênh kết nối này – do đó nó được gọi là kênh dữ liệu. Khi file được truyền xong, kênh này được ngắt. Việc sử dụng các kênh riêng lẻ như vậy tạo ra sự linh hoạt trong việc truyền truyền dữ liệu – mà ta sẽ thấy trong các phần tiếp theo. Tuy nhiên, nó cũng tạo cho FTP độ phức tạp nhất định.

Các tiến trình và thuật ngữ trong FTP

Do các chức năng điều khiển và dữ liệu sử dụng các kênh khác nhau, nên mô hình hoạt động của FTP cũng chia phần mềm trên mỗi thiết bị ra làm hai thành phần logic tương ứng với mỗi kênh. Thành phần Protocol Interpreter (PI) là thành phần quản lý kênh điều khiển, với chức năng phát và nhận lệnh. Thành phần Data Transfer Process (DTP) có chức năng gửi và nhận dữ liệu giữa phía client với server. Ngoài ra, cung cấp cho tiến trình bên phía người dùng còn có thêm thành phần thứ ba là giao diện người dùng FTP - thành phần này không có ở phía server.

Do đó, có hai tiến trình xảy ra ở phía server, và ba tiến trình ở phía client. Các tiến trình này được gắn với mô hình FTP để mô tả chi tiết hoạt động của giao thức FTP. Dưới đây là hình đối chiếu các tiến trình vào trong mô hình FTP:



Các tiến trình phía server:

Các tiến trình phía server bao gồm hai giao thức:

- Server Protocol Interpreter (Server-PI): chịu trách nhiệm quản lý kênh điều khiển trên server. Nó lắng nghe yêu cầu kết nối hướng tới từ users trên cổng dành riêng. Khi kết nối đã được thiết lập, nó sẽ nhận lệnh từ phía User-PI, trả lời lại, và quản lý tiến trình truyền dữ liệu trên server.
- Server DataTransfer Process (Server-DTP): làm nhiệm vụ gửi hoặc nhận file từ bộ phận User-DTP. Server-DTP vừa làm nhiệm thiết lập kết nối kênh dữ liệu và lắng nghe một kết nối kênh dữ liệu từ user. Nó tương tác với server file trên hệ thống cục bộ để đọc và chép file.

Các tiến trình phía client:

- User Protocol Interpreter (User-PI): chịu trách nhiệm quản lý kênh điều khiển phía client. Nó khởi tạo phiên kết nối FTP bằng việc phát ra yêu cầu tới phía Server-PI. Khi kết nối đã được thiết lập, nó xử lý các lệnh nhận được trên giao diện người dùng, gửi chúng tới Server-PI, và nhận phản hồi trở lại. Nó cũng quản lý tiến trình User-DTP.
- User Data Transfer Process (User-DTP): là bộ phận DTP nằm ở phía người dùng, làm nhiệm vụ gửi hoặc nhận dữ liệu từ Server-DTP. User-DTP có thể thiết lập hoặc

lắng nghe yêu cầu kết nối kênh dữ liệu trên server. Nó tương tác với thiết bị lưu trữ file phía client.
- User Interface: cung cấp giao diện xử lý cho người dùng. Nó cho phép sử dụng các lệnh đơn giản hướng người dùng, và cho phép người điều khiển phiên FTP theo dõi được các thông tin và kết quả xảy ra trong tiến trình.

2 - Thiết lập kênh điều khiển và chứng thực người dùng trong FTP:

Mô hình hoạt động của FTP mô tả rõ các kênh dữ liệu và điều khiển được thiết lập giữa FTP client và FTP server. Trước khi kết nối được sử dụng để thực sự truyền file, kênh điều khiển cần phải được thiết lập. Một tiến trình chỉ định sau đó được dùng để tạo kết nối và tạo ra phiên FTP lâu bền giữa các thiết bị để truyền files.

Như trong các giao thức client/server khác, FTP server tuân theo một luật passive trong kênh điều khiển. Bộ phận Server Protocol Interpreter (Server-PI) sẽ lắng nghe cổng TCP dành riêng cho kết nối FTP là cổng 21. Phía User-PI sẽ tạo kết nối bằng việc mở một kết nối TCP từ thiết bị người dùng tới server trên cổng đó. Nó sử dụng một cổng bất kỳ làm cổng nguồn trong phiên kết nối TCP.

Khi TCP đã được cài đặt xong, kênh điều khiển giữa các thiết bị sẽ được thiết lập, cho phép các lệnh được truyền từ User-PI tới Server-PI, và Server-PI sẽ đáp trả kết quả là các mã thông báo. Bước đầu tiên sau khi kênh đã đi vào hoạt động là bước đăng nhập của người dùng (login sequence). Bước này có hai mục đích:
- Access Control - Điều khiển truy cập: quá trình chứng thực cho phép hạn chế truy cập tới server với những người dùng nhất định. Nó cũng cho phép server điều khiển loại truy cập như thế nào đối với từng người dùng.
- Resource Selection - Chọn nguồn cung cấp: Bằng việc nhận dạng người dùng tạo kết nối, FTP server có thể đưa ra quyết định sẽ cung cấp những nguồn nào cho người dùng đã được nhận dạng đó.

Trình tự truy cập và chứng thực FTP

Quy luật chứng thực trong FTP khá đơn giản, chỉ là cung cấp username/password. Trình tự của việc chứng thực như sau:
1 - người dùng gửi một username từ User-PI tới Server-PI bằng lệnh USER. Sau đó password của người dùng được gửi đi bằng lệnh PASS.
2 - Server kiểm tra tên người dùng và password trong database người dùng của nó. Nếu người dùng hợp lệ, server sẽ gửi trả một thông báo tới người dùng rằng phiên kết

nối đã đ.c mở. Nếu người dùng không hợp lệ, server yêu cầu người dùng thực hiện lại việc chứng thực. Sau một số lần chứng thực sai nhất định, server sẽ ngắt kết nối.

Giả sử quá trình chứng thực đã thành công, server sau đó sẽ thiết lập kết nối để cho phép từng loại truy cập đối với người dùng được cấp quyền. Một số người dùng chỉ có thể truy cập vào một số file nhất định, hoặc vào một số loại file nhất định. Một số server có thể cấp quyền cho một số người dùng đọc và viết lên server, trong khi chỉ cho phép đọc đối với những người dùng khác. Người quản trị mạng có thể nhờ đó mà đáp ứng đúng các nhu cầu truy cập FTP.

Một khi kết nối đã được thiết lập, server có thể thực hiện các lựa chọn tài nguyên dựa vào nhận diện người dùng. Ví dụ: trên một hệ thống nhiều người dùng, người quản trị có thể thiết lập FTP để khi có bất cứ người dùng nào kết nối tới, anh ta sẽ tự động được đưa tới "home directory" của chính anh ta. Lệnh tùy chọn ACCT (account) cũng cho phép người dùng chọn một tài khoản cá nhân nào đó nếu như anh ta có nhiều hơn một tài khoản.

Mở rộng về bảo mật FTP

Giống như phần lớn các giao thức cũ, phương pháp đăng nhập đơn giản của FTP là một sự kế thừa từ những giao thức ở thời kỳ đầu của Internet. Ngày nay, nó không còn bảo đảm tính an toàn cần thiết trên môi trường Internet toàn cầu vì username và password được gửi qua kênh kết nối điều khiển dưới dạng clear text. Điều này làm cho các thông tin đăng nhập có thể bị nghe lén. Chuẩn RFC 2228 về các phần mở rộng cho bảo mật FTP đã định ra thêm nhiều tùy chọn chứng thực và mã hóa phức tạp cho những ai muốn tăng thêm mức độ an toàn vào trong phần mềm FTP của họ.

3 - Quản lý kênh dữ liệu FTP, kết nối kênh dữ liệu dạng chủ động (mặc định) và bị động cùng với việc sử dụng cổng

Kênh điều khiển được tạo ra giữa Server-PI và User-PI sử dụng quá trình thiết lập kết nối và chứng thực được duy trì trong suốt phiên kết nối FTP. Các lệnh và các hồi đáp được trao đổi giữa bộ phận PI (Protocol Interpreter) qua kênh điều khiển, nhưng dữ liệu thì không.

Mỗi khi cần phải truyền dữ liệu giữa server và client, một kênh dữ liệu cần phải được tạo ra. Kênh dữ liệu kết nối bộ phận User-DTP với Server-DTP. Kết nối này cần thiết

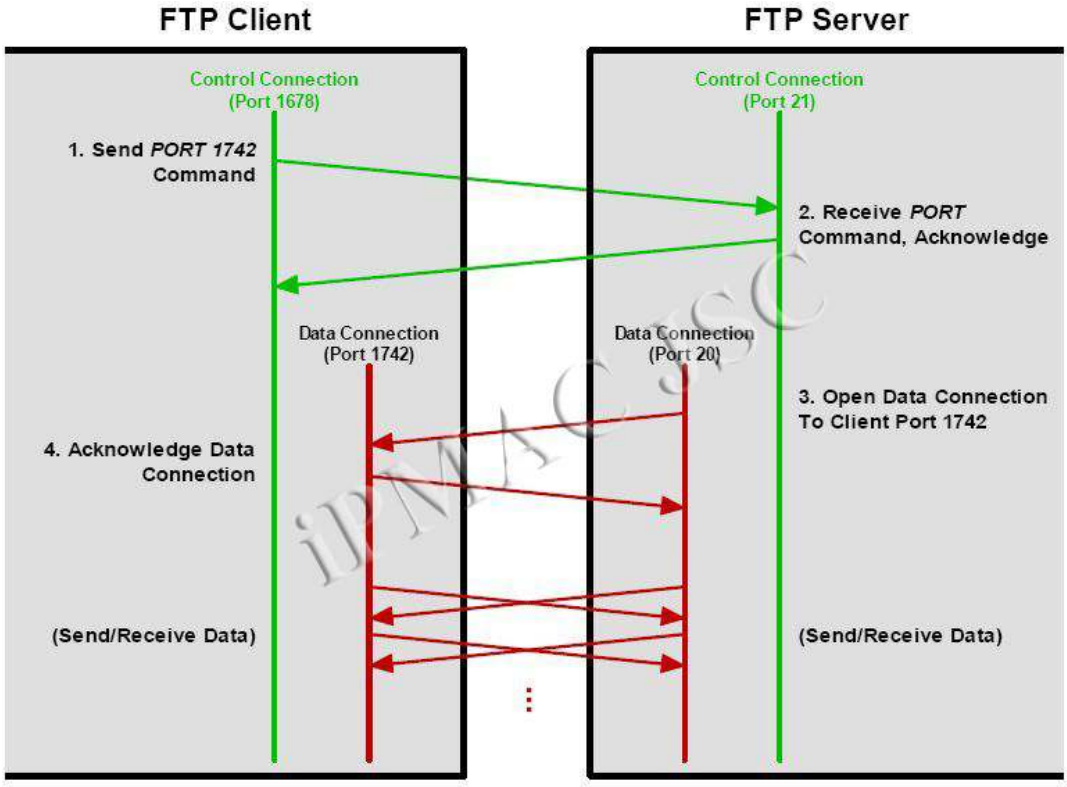
cho cả hoạt động chuyển file trực tiếp (gửi hoặc nhận một file) cũng như đối với việc truyền dữ liệu ngầm, như là yêu cầu một danh sách file trong thư mục nào đó trên server.
Chuẩn FTP chỉ định hai phương thức khác nhau để tạo ra kênh dữ liệu. Khác biệt chính của hai phương thức đó là ở mặt thiết bị: phía client hay phía server là phía đã đưa ra yêu cầu khởi tạo kết nối. Điều này nghe qua có vẻ khá đơn giản, nhưng kỳ thực nó lại khá quan trọng.

Kết nối kênh dữ liệu dạng chủ động

Phương thức đầu tiên đôi khi còn được gọi là kết nối kênh dữ liệu dạng thông thường (vì nó là phương pháp mặc định) và đôi khi được gọi là kết nối dạng chủ động (để đối chiếu với dạng kết nối bị động mà ta sẽ xét ở phần sau). Trong dạng kết nối này, phía Server-DTP khởi tạo kệnh dữ liệu bằng việc mở một cổng TCP cho phía User-DTP. Phía server sử dụng cổng được dành riêng, là cổng 20 cho kênh dữ liệu. Trên máy client, một giá trị cổng được chọn theo mặc định chính là cổng được sử dụng đối với kênh điều khiển, tuy nhiên phía client sẽ luôn chọn hai cổng riêng biệt cho hai kênh này.

Giả sử phía User-PI thiết lập một kết nối điều khiển từ cổng bất kỳ của nó là 1678 tới cổng điều khiển trên server là cổng 21. Khi đó, để tạo một kênh dữ liệu cho việc truyền dữ liệu, phía Server-PI sẽ báo cho phía Server-DTP khởi tạo một kênh kết nối TCP từ cổng 20 tới cổng 1678 của phía client. Sau khi phía client chấp nhận kênh được khởi tạo, dữ liệu sẽ được truyền đi.

Thực tế, việc sử dụng cùng một cổng cho cả kênh dữ liệu và kênh điều khiển không phải là một ý hay, nó làm cho hoạt động của FTP trở nên phức tạp. Do đó, phía client nên chỉ định sử dụng một cổng khác bằng việc sử dụng lệnh PORT trước khi truyền dữ liệu. Ví dụ: giả sử phía client chỉ định cổng 1742 với lệnh PORT. Phía Server-DTP sau đó sẽ tạo ra một kết nối từ cổng 20 của nó tới cổng 1742 phía client thay vì cổng 1678 như mặc định. Quá trình này được mô tả trong hình dưới đây.



Thông thường, đối với kênh dữ liệu FTP, phía server sẽ khởi tạo việc truyền dữ liệu bằng cách mở kết nối dữ liệu tới client.

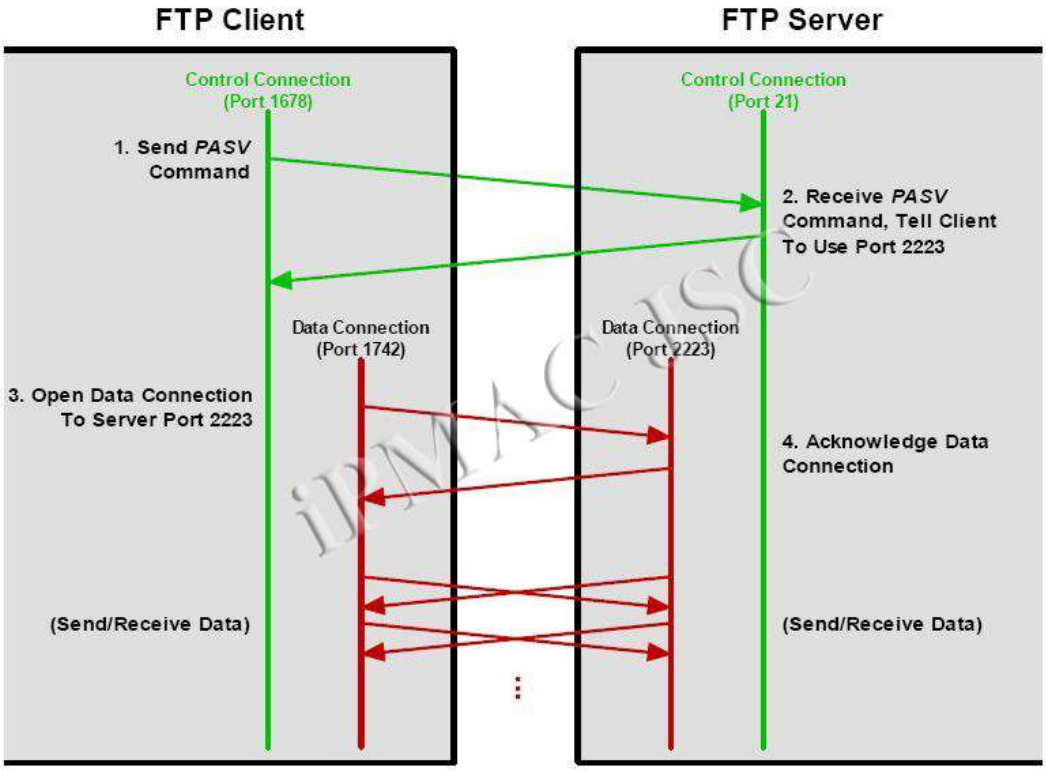
Trong trường hợp trên, phía client trước tiên sẽ đưa ra lệnh PORT để yêu cầu server sử dụng cổng 1742. Sau đó, server sẽ mở kết nối kênh dữ liệu từ cổng 20 mặc định của nó tới cổng 1742 phía client. Dữ liệu sau đó sẽ được truyền giữa các thiết bị qua các cổng này.

Kết nối kênh dữ liệu dạng bị động

Phương pháp kế tiếp được gọi là kết nối dữ liệu dạng bị động. Phía client sẽ nhận server là phía bị động, làm nhiệm vụ chấp nhận một yêu cầu kết nối kênh dữ liệu được khởi tạo từ phía client. Server trả lời lại phía client với địa chỉ IP cũng như địa chỉ cổng mà nó sẽ sử dụng. Phía Server-DTP sau đó sẽ lắng nghe một kết nối TCP từ phía User-DTP trên cổng này.

Mặc định, phía client sử dụng cùng một cổng đối với cả hai kênh điều khiển và dữ liệu như trong trường hợp kết nối chủ động ở trên. Tuy nhiên, ở đây, một lần nữa phía client có thể chọn sử dụng một giá trị cổng khác cho kênh dữ liệu. Ta sẽ xét lại ví dụ ở trên một lần nữa, với cổng điều khiển phía client là 1678 tới cổng 21 phía server.

Nhưng lần này truyền dữ liệu theo phương thức kết nối bị động, như mô tả trong hình dưới đây:



Phía client sẽ sử dụng lệnh PASV để yêu cầu server rằng nó muốn dùng phương thức điều khiển dữ liệu bị động. Phía Server-PI sẽ trả lời lại phía client với một giá trị cổng mà client sẽ sử dụng, từ cổng 2223 trên nó. Sau đó phía Server PI sẽ hướng cho phía Server-DTP lắng nghe trên cổng 2223. Phía User-PI cũng sẽ hướng cho phía User-DTP tạo một phiên kết nối từ cổng 1742 phía client tới cổng 2223 phía server. Sau khi Server chấp nhận kết nối này, dữ liệu bắt đầu được truyền đi.

Các vấn đề về tính hiệu quả và tính bảo mật trong việc chọn một phương thức kết nối

Vấn đề phía nào là phía khởi tạo kết nối kênh dữ liệu đưa ra một câu hỏi: sự khác nhau giữa hai phương thức là gì? Điều này cũng giống như việc hỏi ai đã thực hiện một cuộc điện thoại nội bộ. Câu trả lời là sự bảo mật. Việc FTP sử dụng nhiều hơn một kết nối TCP có thể giải quyết các vấn đề về phần mềm cũng như về phần cứng mà người dùng cần phải có để đảm bảo sự an toàn cho hệ thống của họ.

Khi xem xét việc gì sẽ xảy ra trong trường hợp kênh dữ liệu chủ động như trong ví dụ phía trên:

Đối với phía client, có một kênh kết nối điều khiển được thiết lập từ cổng 1678 client tới cổng 21 server. Nhưng kênh dữ liệu lại được khởi tạo từ phía server. Do đó, client sẽ nhận được một yêu cầu kết nối tới cổng 1678 (hoặc cổng nào khác). Một số client sẽ nghi ngờ về việc nhận được những kết nối tới như vậy, vì trong tình huống thông thường, client mới là phía khởi tạo kết nối chứ không phải đáp trả kết nối. Do các kênh kết nối TCP hướng tới có thể mang theo những mối đe dọa nhất định, một số client có thể sẽ ngăn chặn các luồng kết nối hướng tới bằng việc sử dụng tường lửa.

Tại sao người ta lại không làm cho phía client luôn chấp nhận kết nối từ một chỉ số port được dùng trong kênh điều khiển? Vấn đề ở đây là vì client thường dùng các cổng khác nhau cho mỗi phiên kết nối bằng việc sử dụng câu lệnh PORT. Và tại sao điều này lại được thực hiện? Vì theo luật TCP: sau khi một kết nối được đóng lại , có một khoảng thời gian trống trước khi cổng đó có thể được sử dụng lại – điều này để ngăn ngừa tình trạng các phiên kết nối liên tiếp bị lẫn với nhau. Điều này sẽ tạo ra độ trễ khi gửi nhiều file – do đó phía client thường dùng các giá trị cổng khác nhau cho mỗi kết nối. Điều này rất hiệu quả nhưng cũng dẫn tới việc firewall của client sẽ hỏi có chấp nhận phiên kết nối tới với nhiều giá trị cổng không ổn định hay không.

Việc dùng kết nối kiểu kênh gián tiếp sẽ giảm thiểu vấn đề này một cách hiệu quả. Phần lớn các tường lửa có nhiều vấn đề liên quan tới kết nối hướng về với các giá trị cổng bất kỳ, hơn là gặp vấn đề với các kết nối hướng đi. Ta có thể xem chi tiết hơn về vấn đề này trong chuẩn RFC 1579. Chuẩn này khuyến nghị rằng phía client nên sử dụng kết nối kiểu bị động làm dạng mặc định thay vì sử dụng kiểu kết nối dạng chủ động cùng với lệnh PORT, để ngăn chặn tình trạng block theo cổng. Tất nhiên, phương thức kết nối kiểu bị động không hoàn toàn giải quyết được vấn đề, chúng chỉ đẩy vấn đề về phía server mà thôi. Phía server, giờ đây phải đối mặt với việc có nhiều kênh kết nối hướng về trên hàng loạt các cổng khác nhau. Tuy nhiên việc xử lý các vấn đề bảo mật trên một nhóm nhỏ server vẫn dễ hơn nhiều so với việc phải đối mặt với một lượng lớn các vấn đề từ nhiều client. FTP server phải được cấu hình chấp nhận phương thức truyền bị động từ client, do đó cách thông thường để thiết lập trên server là thiết lập chấp nhận một số cổng kết nối hướng về trên server trong khi vẫn khóa các yêu cầu kết nối hướng về trên các cổng khác.

4 - Các phương thức truyền dữ liệu trong FTP

Khi kênh dữ liệu đã được thiết lập xong giữa Server-DTP với User-DTP, dữ liệu sẽ được truyền trực tiếp từ phía client tới phía server, hoặc ngược lại, dựa theo các lệnh

được sử dụng. Do thông tin điều khiển được gửi đi trên kênh điều khiển, nên toàn bộ kênh dữ liệu có thể được sử dụng để truyền dữ liệu. (Tất nhiên, hai kênh logic này được kết hợp với nhau ở lớp dưới cùng với tất cả các kết nối TCP/UDP khác giữa hai thiết bị, do đó điều này không hẳn đã cải thiện tốc độ truyền dữ liệu so với khi truyền trên chỉ một kênh – nó chỉ làm cho hai việc truyền dữ liệu và điều khiển trở nên độc lập với nhau mà thôi)
FTP có ba phương thức truyền dữ liệu, nêu lên cách mà dữ liệu được truyền từ một thiết bị tới thiết bị khác trên một kênh dữ liệu đã được khởi tạo, đó là: stream mode, block mode, và compressed mode

Stream mode

Trong phương thức này, dữ liệu được truyền đi dưới dạng các byte không cấu trúc liên tiếp. Thiết bị gửi chỉ đơn thuần đầy luồng dữ liệu qua kết nối TCP tới phía nhận. Không có một trường tiêu đề nhất định được sử dụng trong phương thức này làm cho nó khá khác so với nhiều giao thức gửi dữ liệu rời rạc khác. Phương thức này chủ yếu dựa vào tính tin cậy trong truyền dữ liệu của TCP. Do nó không có cầu trúc dạng header, nên việc báo hiệu kết thúc file sẽ đơn giản được thực hiện việc phía thiết bị gửi ngắt kênh kết nối dữ liệu khi đã truyền xong.
Trong số ba phương thưc, stream mode là phương thức được sử dụng nhiều nhất trong triển khai FTP thực tế. Có một số lý do giải thích điều đó. Trước hết, nó là phương thức mặc định và đơn giản nhất, do đó việc triển khai nó là dễ dàng nhất. Thứ hai, nó là phương pháp phổ biến nhất, vì nó xử lý với các file đều đơn thuần như là xử lý dòng byte, mà không để ý tới nội dung của các file. Thứ ba, nó là phương thức hiệu quả nhất vì nó không tốn một lượng byte "overload" để thông báo header.

Block mode

Đây là phương thức truyền dữ liệu mang tính quy chuẩn hơn, với việc dữ liệu được chia thành nhiều khối nhỏ và được đóng gói thành các FTP blocks. Mỗi block này có một trường header 3 byte báo hiệu độ dài, và chứa thông tin về các khối dữ liệu đang được gửi. Một thuật toán đặc biệt được sử dụng để kiểm tra các dữ liệu đã được truyền đi và để phát hiện, khởi tạo lại đối với một phiên truyền dữ liệu đã bị ngắt.

Compressed mode

Đây là một phương thức truyền sử dụng một kỹ thuật nén khá đơn giản, là "run-length encoding" – có tác dụng phát hiện và xử lý các đoạn lặp trong dữ liệu được truyền đi

để giảm chiều dài của toàn bộ thông điệp. Thông tin khi đã được nén, sẽ được xử lý như trong block mode, với trường header. Trong thực tế, việc nến dữ liệu thường được sử dụng ở những chỗ khác, làm cho phương thức truyền kiểu compressed mode trở nên không cần thiết nữa. Ví dụ: nếu bạn đang truyền đi một file qua internet với modem tương tự, modem của bạn thông thường sẽ thực hiện việc nén ở lớp 1; các file lớn trên FTP server cũng thường được nén sẵn với một số định dạng như ZIP, làm cho việc nén tiếp tục khi truyền dữ liệu trở nên không cần thiết.

3.4.2. Cài đặt FTP Client/Server

Trên cơ sở giao thức FTP chúng ta thực hiện cài đặt FTP Client/Server để minh họa cho giao thức

Chương trình Simple FTP Server:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
class Program {
   static void Main(string[] args) {
      string rootDir = "C:/MyFTP";
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2121);
      TcpListener server = new TcpListener(iep);
      server.Start();
      TcpClient client = server.AcceptTcpClient();
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      sw.WriteLine("220 Chao mung ket noi toi MyFTP");
      sw.Flush();
      while (true) {
         string request = sr.ReadLine();
         string command="";
         if(request.Length!=0)         command  = request.Substring(0, 4);
         switch (command.ToUpper().Trim()) {
            case "USER": {
                  sw.WriteLine("331. Nhap pass vao");
                  sw.Flush();
                  //sw.Close();
                  Console.WriteLine(request);
                  break;
               }
            case "PASS": {
                  sw.WriteLine("230. Dang nhap thanh cong");
                  sw.Flush();
                  Console.WriteLine(request);
                  break;
               }
```

```
            case "MKD": {
                string folderName = request.Substring(4, request.Length - 4);
                folderName = rootDir + "/" + folderName.Trim();
                try {
                   Directory.CreateDirectory(folderName);
                   sw.WriteLine("150 Tao thu muc thanh cong");
                   sw.Flush();
                }catch(IOException){
                   sw.WriteLine("550 Tao thu muc co loi");
                   sw.Flush();
                }
                break;
              }
            case "RETR": {
                string fileName = request.Substring(4, request.Length - 4);
                fileName = rootDir + "/" + fileName.Trim();
                try {
                   if (File.Exists(fileName)) {
                      //Gui noi dung file ve cho client xu ly
                      sw.WriteLine("150 Truyen File thanh cong");
                      sw.Flush();
                      FileStream fs = new FileStream(fileName, FileMode.Open);
                      long totalLenght = fs.Length;
                      byte[] data = new byte[totalLenght];
                      fs.Read(data, 0, data.Length);
                      sw.Write(totalLenght);
                      char[] kt = Encoding.ASCII.GetChars(data);
                      sw.Write(kt,0,data.Length);
                      sw.Flush();
                      fs.Close();
                   } else {
                      sw.WriteLine("550 File khong ton tai tren server");
                      sw.Flush();
                   }
                } catch (IOException) {
                   sw.WriteLine("550 Khong truyen duoc file");
                   sw.Flush();
                }
                break;
              }
            case "STOR": {
                string fileName =
request.Substring(request.LastIndexOf("/"),request.Length-request.LastIndexOf("/"));
                fileName = rootDir + "/" + fileName.Trim();
                try {
                   FileStream fs = new FileStream(fileName, FileMode.CreateNew);
                   long totalLength=sr.Read();
                   byte[] data = new byte[totalLength];
                   char[] kt = Encoding.ASCII.GetChars(data);
```

```
                    int sobyte = sr.Read(kt, 0, data.Length);
                    fs.Write(data, 0, sobyte);
                    fs.Close();
                    sw.WriteLine("150 Up file thanh cong");
                    sw.Flush();
                } catch (IOException) {
                    sw.WriteLine("550 Khong truyen duoc file");
                    sw.Flush();
                }
                break;
            }
        case "QUIT": {
                client.Close();
                break;
            }
        default: {
                sw.WriteLine("Sai lenh");
                sw.Flush();
                break;
            }
      }
    }
  }
}
```

Chương trình Simple FTP Client:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace FtpClient {
  class Program {
    static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2121);
      TcpClient client = new TcpClient();
      client.Connect(iep);
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      Console.WriteLine(sr.ReadLine());
      string input;
      string command = "";
      Console.WriteLine("Dang nhap bang USER Ten user, PASS Ten password");
      Console.WriteLine("Tao thu muc bang MKD ten thu muc can tao");
      Console.WriteLine("Upload bang cach STOR tenfile");
      Console.WriteLine("Download bang cach RETR tenfile");
      while (true) {
        input = Console.ReadLine();
```

```
command = input.Substring(0, 4).Trim().ToUpper();
switch (command) {
   case "STOR": {
         //Doc file gui cho server
         sw.WriteLine(input);
         sw.Flush();
         FileInfo fl=null;
         try {
            fl = new FileInfo(input.Substring(4, input.Length - 4).Trim());
         } catch (IOException) {
            Console.WriteLine("File khong ton tai");
         }
         long totalLength = fl.Length;
         FileStream fs = fl.OpenRead();
         sw.Write(totalLength);
         byte[] data = new byte[totalLength];
         int bytes = fs.Read(data, 0, data.Length);
         char[] kt = Encoding.ASCII.GetChars(data);
         sw.Write(kt, 0, data.Length);
         sw.Flush();
         fs.Close();
         Console.WriteLine(sr.ReadLine());
         break;
      }
   case "RETR": {
         sw.WriteLine(input);
         sw.Flush();
         string s = sr.ReadLine();
         Console.WriteLine(s);
         if (s.Substring(0, 3).Equals("150")) {
            Console.Write("Nhap vao noi luu tep:");
            string filename = Console.ReadLine();
            FileStream fs = new FileStream(filename, FileMode.CreateNew);
            //Doc tep ve;
            long totalLength = sr.Read();
            byte[] data = new byte[totalLength];
            char[] kt= new char[data.Length] ;
            int sobyte = sr.Read(kt, 0, data.Length);
            data=Encoding.ASCII.GetBytes(kt);
            fs.Write(data, 0, data.Length);
            fs.Close();
         }
         break;
      }
   default: {
         sw.WriteLine(input);
         sw.Flush();
         Console.WriteLine(sr.ReadLine());
         break;
```

```
                }
            }
            if (input.ToUpper().Equals("QUIT")) break;
        }
        sr.Close();
        sw.Close();
        client.Close();
    }
  }
}
```

3.5. DNS (Domain Name Server)

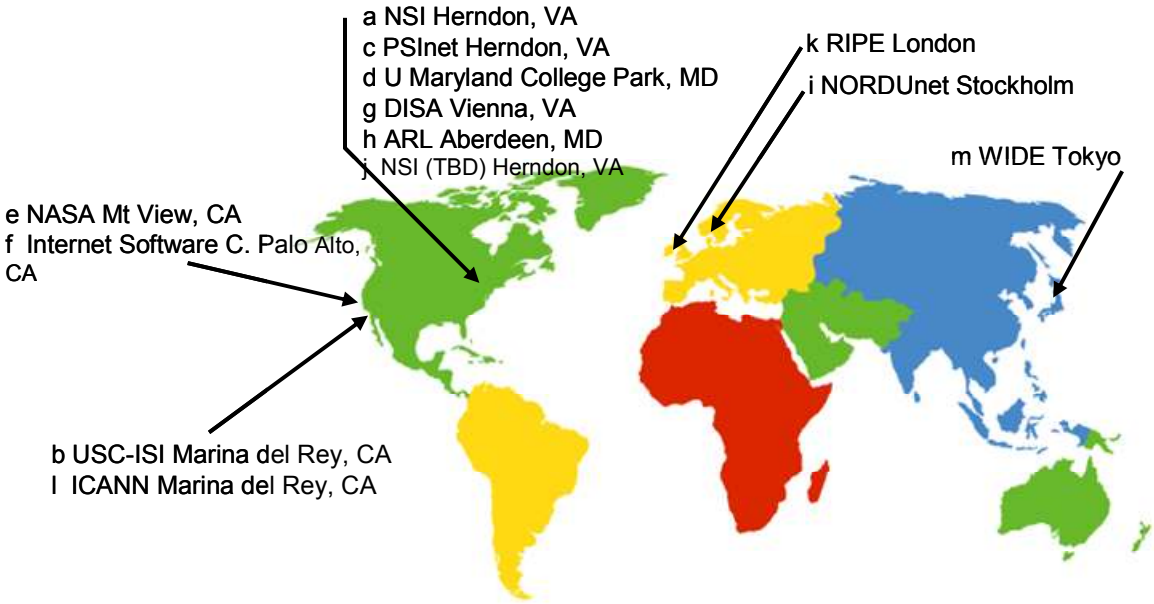
3.5.1. Vấn đề phân giải tên miền

Domain Name System:

- *Là hệ cơ sở dữ liệu phân tán hoạt động có thứ bậc bởi các name servers*
- *Là giao thức tầng ứng dụng :* host, routers yêu cầu tới name servers để xác định tên miền (ánh xạ địa chỉ <->tên miền)
  - Note : là một chức năng của Internet, hoạt động như là giao thức tầng ứng dụng
  - Rất phức tạp.

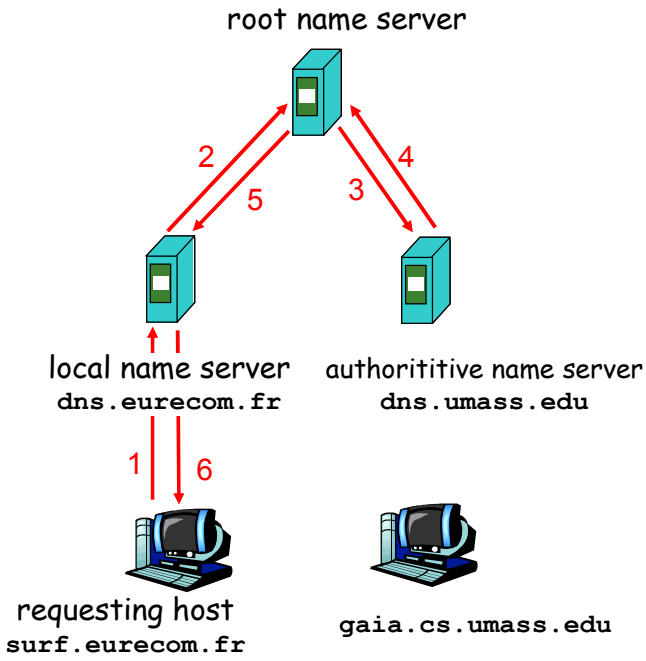
Q: Ánh xạ giữa địa chỉ IP và tên?

- **Tại sao không tập trung sự kiểm soát của DNS ?**
  - Điểm hỏng duy nhất - nếu name-server "chết" thì cả mạng Internet sẽ "chết" theo.
  - Tốn đường truyền.
  - Cơ sở dữ liệu tập trung sẽ "xa" với đa số vùng
  - Bảo trì phức tạp.
  - Phải chia để trị *!*
  - Không có server nào có thể lưu toàn bộ được tên miền và địa chỉ IP tương ứng
- **local name servers:**
  - Mỗi ISP,công ty có *local (default) name server*
  - Câu hỏi truy vấn của host về DNS sẽ được chuyển tới local name server
- **Chức năng của name server:**
  - Đối với host: lưu địa chỉ IP và tên miền tương ứng của host
  - Có thể tìm tên miền ứng với địa chỉ IP và ngược lại
- Được yêu cầu bởi các local name server không thể xác định được tên.
- root name server:
  - Được yêu cầu nếu có authoritative name server không xác định.
  - Nhận và xử lý mapping
  - Trả về mapping cho local name server



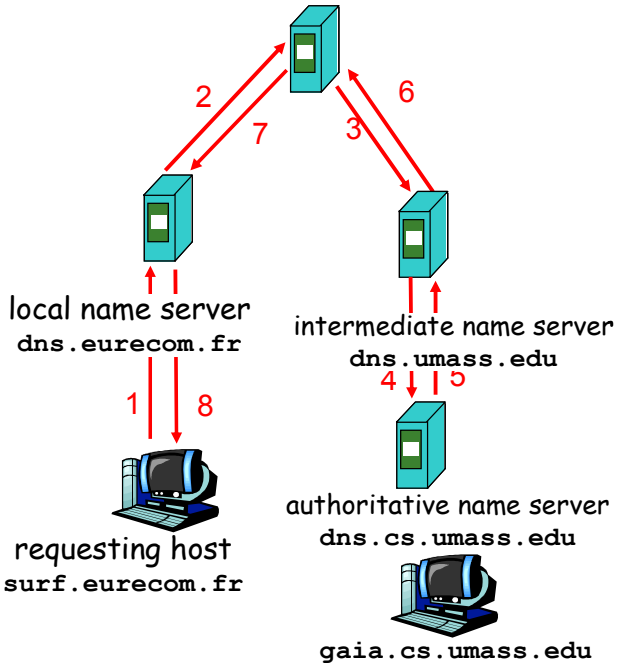
host **surf.eurecom.fr** muốn biết địa chỉ IP của **gaia.cs.umass.edu**

1. Yêu cầu tới local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** yêu cầu tới root name server nếu cần thiết
3. root name server yêu cầu authoritative name server, **dns.umass.edu,** nếu cần thiết.



Root name server:

- Có thể không biết authoritative name server
- Có thể biết *name server trung gian* ,nhờ đó có thể yêu cầu tìm authoritative name server



DNS example

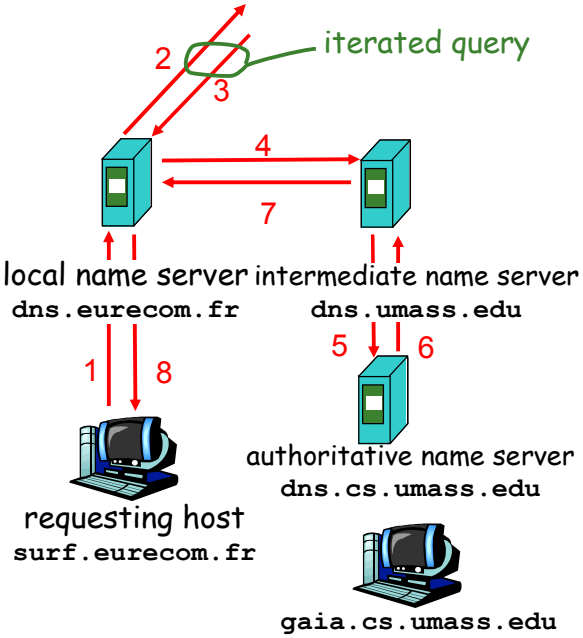
- Truy vấn trong DNS được chia thành các loại như sau:

Truy vấn đệ quy query:

- Name server là nơi phân gi ải địa chỉ/tên.Nếu nó không phân giải trong nội bộ,nó sẽ gửi yêu cầu đến name server khác.
- Công việc của name server liệu có quá nặng?

Truy vấn tương tác:

- Nếu không phân giải được địa chỉ IP/name,name server sẽ gửi trả thông đi ệp rằng "Tôi không biết,hãy thử hỏi anh bạn cạnh tôi là A".A là đ ịa chỉ IP của name server kế tiếp nó.



- Cấu trúc bản ghi DNS như sau:

DNS: cơ sở dữ liệu phân tán lưu các bản ghi nguồn (RR)

Định dạng của RR : **(name, value, type, ttl)**

- **Type=A**
  - **name** : hostname
  - **value** : IP address
- **Type=NS**
  - **name** : domain (e.g. foo.com)
  - **value** : địa chỉ IP authoritative name server cho tên miền đó
- **Type=CNAME**
  - **name** : tên bí danh cho một tên thực nào đó : e.g www.ibm.com là tên bí danh của servereast.backup2.ibm.com
  - **value** : là tên thực
- **Type=MX**
  - **value** : tên của mailserver

3.5.2. Triển khai DNS MX (Mail Exchange)

Chúng ta đi viết chương trình cho phép lấy về thông tin của mail server

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
public partial class Form1 : Form {
   public Form1() {
      InitializeComponent();
   }
   private void btFind_Click(object sender, EventArgs e) {
      byte[] DNSQuery;
      byte[] DNSReply;
      UdpClient dnsClient = new UdpClient(tbServer.Text, 53);
      DNSQuery = makeQuery(DateTime.Now.Millisecond *
      60, tbDomain.Text);
      dnsClient.Send(DNSQuery, DNSQuery.GetLength(0));
      IPEndPoint endpoint = null;
      DNSReply = dnsClient.Receive(ref endpoint);
      this.tbStatus.Text = makeResponse(DNSReply, tbDomain.Text);
   }
   public byte[] makeQuery(int id, string name) {
      byte[] data = new byte[512];
      byte[] Query;
      data[0] = (byte)(id >> 8);
      data[1] = (byte)(id & 0xFF);
      data[2] = (byte)1; data[3] = (byte)0;
      data[4] = (byte)0; data[5] = (byte)1;
```

```
        data[6] = (byte)0; data[7] = (byte)0;
        data[8] = (byte)0; data[9] = (byte)0;
        data[10] = (byte)0; data[11] = (byte)0;
        string[] tokens = name.Split(new char[] { '.' });
        string label;
        int position = 12;
        for (int j = 0; j < tokens.Length; j++) {
           label = tokens[j];
           data[position++] = (byte)(label.Length & 0xFF);
           byte[] b = System.Text.Encoding.ASCII.GetBytes(label);
           for (int k = 0; k < b.Length; k++) {
              data[position++] = b[k];
           }
        }
        data[position++] = (byte)0; data[position++] = (byte)0;
        data[position++] = (byte)15; data[position++] = (byte)0;
        data[position++] = (byte)1;
        Query = new byte[position + 1];
        for (int i = 0; i <= position; i++) {
           Query[i] = data[i];
        }
        return Query;
     }
     public string makeResponse(byte[] data, string name) {
        int qCount = ((data[4] & 0xFF) << 8) | (data[5] & 0xFF); int aCount = ((data[6]
& 0xFF) << 8) | (data[7] & 0xFF);
        int position = 12;
        for (int i = 0; i < qCount; ++i) {
           name = "";
           position = proc(position, data, ref name);
           position += 4;
        }
        string Response = "";
        for (int i = 0; i < aCount; ++i) {
           name = "";
           position = proc(position, data, ref name);
           position += 12;
           name = "";
           position = proc(position, data, ref name);
           Response += name + "\r\n";
        }
        return Response;
     }
     private int proc(int position, byte[] data, ref string name) {
        int len = (data[position++] & 0xFF);
        if (len == 0) {
           return position;
        }
        int offset;
```

```
        do {
            if ((len & 0xC0) == 0xC0) {
                if (position >= data.GetLength(0)) {
                    return -1;
                }
                offset = ((len & 0x3F) << 8) | (data[position++] &
                  0xFF);
                proc(offset, data, ref name);
                return position;
            } else {
                if ((position + len) > data.GetLength(0)) {
                    return -1;
                }
                name += Encoding.ASCII.GetString(data, position, len);
                position += len;
            }
            if (position > data.GetLength(0)) {
                return -1;
            }
            len = data[position++] & 0xFF;
            if (len != 0) {
                name += ".";
            }
        }
        while (len != 0);
        return position;
    }
}
```

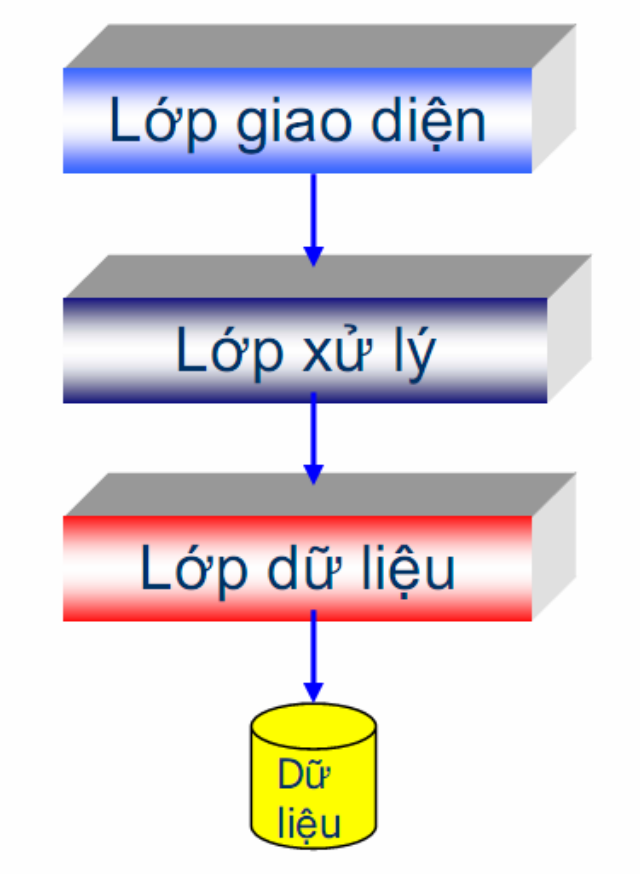
3.6 Thảo luận về các ứng dụng khác thường gặp

3.7 Bài tập áp dụng

CHƯƠNG 4: XÂY DỰNG ỨNG DỤNG NHIỀU LỚP

4.1. Mô hình 2 lớp (two tier), 3 lớp (three tier) và n lớp.

Trước đây, đối với các phần mềm có sử dụng liên quan đến dữ liệu, thường khi làm người lập trình thường tích hợp việc giao tiếp với người sử dụng , xử lý rồi ghi xuống dữ liệu trên cùng một form (đây là mô hình một lớp). Nhưng trong kiến trúc 3 lớp (mô hình 3 lớp), phải có việc phân biệt rạch ròi giữa các lớp này. Mô hình 3 lớp có thể được mô tả như sau:



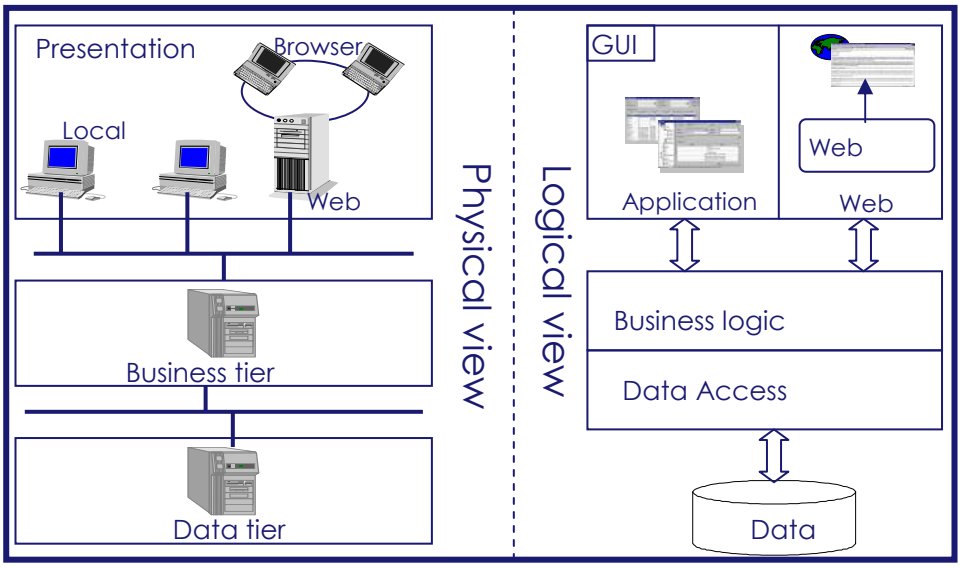
- Lớp thứ nhất : Lớp giao diện (giao tiếp với người sử dụng) : chỉ thuần xử lý việc giao tiếp với người sử dụng, nhập xuất, … mà không thực hiện việc tính toán, kiểm tra, xử lý, hay các thao tác liên quan đến cơ sở dữ liệu.
- Lớp thứ hai : Lớp xử lý : Lớp này chuyên thực hiện các xử lý , kiểm tra các ràng buộc, các qui tắc ứng xử của phần mềm , các chức năng cốt yếu, … Việc thực hiện này độc lập với cách thiết kế cũng như cài đặt giao diện. Thông tin cho lớp này thực hiện các xử lý của mình được lấy từ lớp giao diện.
- Lớp thứ ba : Lớp dữ liệu : Lớp này chuyên thực hiện các công việc liên quan đến dữ liệu. Dữ liệu có thể lấy từ cơ sở dữ liệu (Access, SQL Server …) hoặc tập tin (text, binary, XML …). Đối với cơ sở dữ liệu, lớp này thực hiện kết nối trực tiếp với cơ sở dữ liệu và thực hiện tất cả các thao tác liên quan đến cơ sở dữ liệu mà phần mềm cần thiết. Đối với tập tin, lớp này thực hiện việc đọc, ghi tập tin theo yêu cầu của phần mềm. Việc thực hiện này do lớp xử lý gọi.

Rõ ràng, với mô hình này, các công việc của từng lớp là độc lập với nhau. Việc thay đổi ở một lớp không làm thay đổi các lớp còn lại, thuận tiện hơn cho quá trình phát triển và bảo trì phần mềm.

Một số lưu ý:

- Phân biệt vai trò Business Layer và khái niệm "xử lý"
- Mỗi Layer vẫn có xử lý riêng, đặc trưng của Layer đó
- Đôi khi việc quyết định 1 xử lý nằm ở layer nào chỉ mang tính chất tương đối

Chúng ta cũng cần phân biệt khái niệm 3 tier và 3 layer: 3 tier là mô hình 3 lớp vật lý còn 3 layer là mô hình logic.

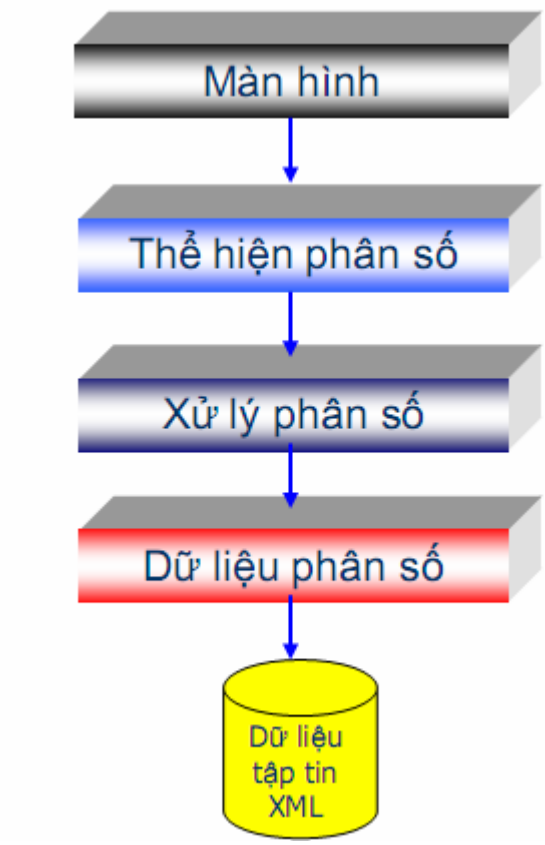


Ví dụ minh họa:

Xây dựng chương trình tính tổng 2 phân số theo kiến trúc 3 lớp. Theo đó dữ liệu của phân số được đọc lên từ tập tin XML, kết quả sau khi được tính sẽ được ghi xuống tập tin XML.

Cách làm thông thường là mọi việc đều được đẩy vào trong 1 form và xử lý trực tiếp trong form đó. Tuy nhiên, khi có sự thay đổi xảy ra về giao diện, xử lý, hay dữ liệu thì việc chỉnh sửa khá khó khăn. Do vậy, việc xây dựng theo kiến trúc 3 lớp sẽ khắc phục nhược điểm này.

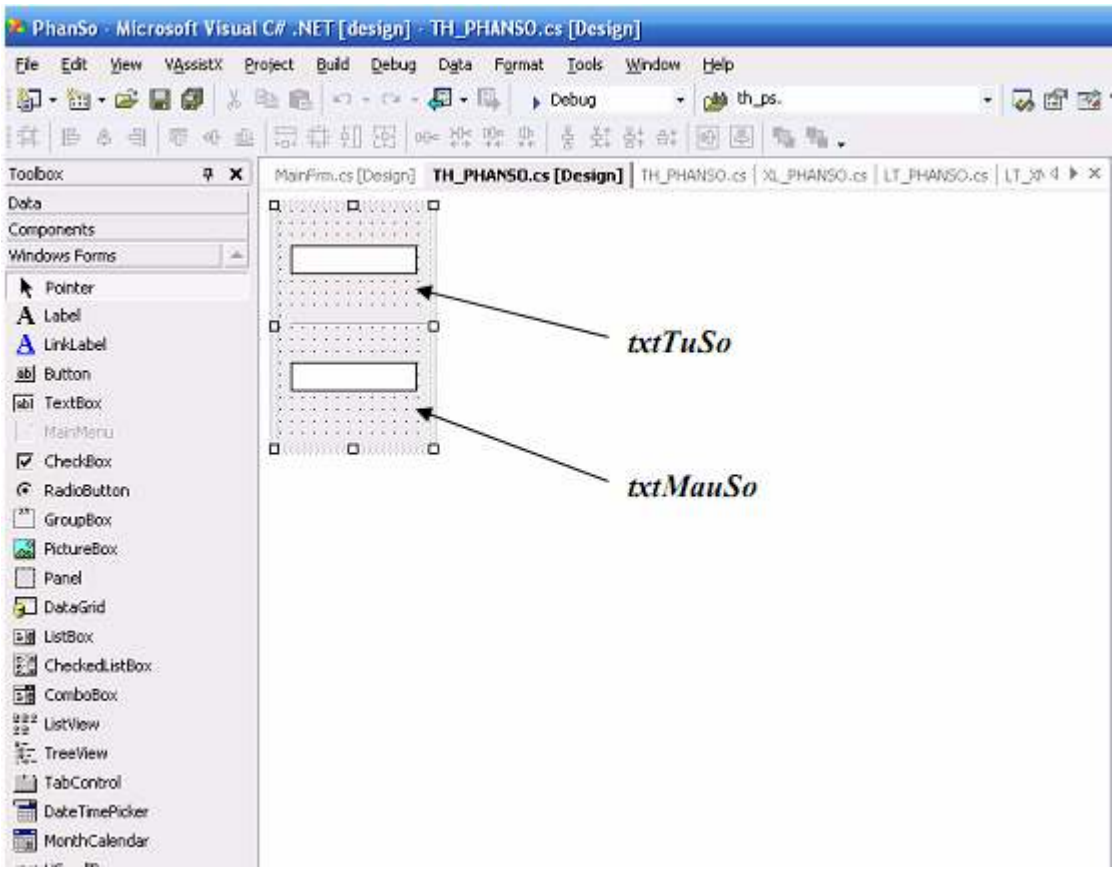
Kiến trúc của chương trình như sau:



Xây dựng lớp thể hiện phân số (TH_PHANSO)

Sử dụng User Control để cài đặt cho TH_PHANSO. Thêm User Control vào project bằng cách chọn Project > Add User Control. Đặt tên User Control đó.

Ta có TH_PHANSO.cs



Do thể hiện tử số và thể hiện mẫu số đều là TextBox do đó trong lớp TH_PHANSO cần thiết lập các properties là tuso và mauso có kiểu int.

```
public int tuso{
  set{
   this.txtTuSo.Text = value.ToString();
 }
  get{
   return int.Parse(this.txtTuSo.Text);
 }
}
 public int mauso {
  set  {
   this.txtMauSo.Text = value.ToString();
 }
  get  {
   return int.Parse(this.txtMauSo.Text);
 }
}
```

Lớp lưu trữ phân số (LT_PHANSO)

Tập tin XML lưu trữ có định dạng như sau:

```
<?xml version ="1.0" encoding = "utf-8"?>
<PHANSO>
 <Tu_so>5</Tu_so>
 <Mau_so>3</Mau_so>
</PHANSO>
```

Để thực hiện việc đọc và ghi dữ liệu XML ta sử dụng DOM.

Khai báo tuso và mauso để thực hiện việc lưu trữ

public int tuso;

public int mauso;

Thực hiện cài đặt hàm khởi tạo mặc định với tham số truyền vào là đường dẫn file XML

```
public LT_PHANSO(string strFilename)
{
  //
  // TODO: Add constructor logic here
  //
  XmlDocument doc = LT_XML.DocTaiLieu(strFilename);
  if(doc == null)
  {
        tuso = 0;
```

```
        mauso = 0;
        return;
 }
 XmlElement ele = doc.DocumentElement;
 tuso = int.Parse(ele.SelectSingleNode("Tu_so").InnerText);
 mauso = int.Parse(ele.SelectSingleNode("Mau_so").InnerText);
}
```

Thực hiện cài đặt hàm ghi phân số với tham số truyền vào là đường dẫn file XML

```
public void GhiPhanSo(string strFilename)
{
  XmlDocument doc = new XmlDocument();
  XmlElement root = doc.CreateElement("PHANSO");
 doc.AppendChild(root);
  XmlElement ele_Tuso =
root.OwnerDocument.CreateElement("Tu_so");
 ele_Tuso.InnerText = this.tuso.ToString();
 root.AppendChild(ele_Tuso);
  XmlElement ele_Mauso =
root.OwnerDocument.CreateElement("Mau_so");
 ele_Mauso.InnerText = this.mauso.ToString();
 root.AppendChild(ele_Mauso);
  LT_XML.GhiTaiLieu(strFilename,doc);
}
```

Lớp lưu trữ XML (LT_XML)

Việc load và save XmlDocument được tách ra thành một lớp riêng là lớp LT_XML

```
public static XmlDocument DocTaiLieu(string strFilename) {
  XmlDocument kq = new XmlDocument();
  try  {
         kq.Load(strFilename);
 }
  catch{
         return null;
 }
  return kq;
}
 public static void GhiTaiLieu(string strFilename, XmlDocument doc) {
  try{
           doc.Save(strFilename);
 }
  catch{
 }
}
```

Lớp xử lý phân số (XL_PHANSO)

Lớp này sẽ thực hiện cài đặt các hàm liên quan đến xử lý và tính toán trên phân số như định nghĩa phép cộng 2 phân số, rút gọn phân số hay cập nhật giá trị từ đối tượng thể hiện.

Khai báo 2 đối tượng lần lượt thuộc về lớp LT_PHANSO và TH_PHANSO để giúp tạo liên kết với tầng xử lý với 2 tầng còn lại là tầng dữ liệu và tầng giao diện.

```
private LT_PHANSO lt_ps = null;
private TH_PHANSO th_ps = null;
```

Cài đặt hàm khởi tạo mặc định để tạo liên kết với đối tượng thể hiện và đối tượng xử lý

```
public XL_PHANSO(LT_PHANSO lt_ps, TH_PHANSO th_ps)
{
  this.lt_ps = lt_ps;
     this.th_ps = th_ps;
  this.th_ps.tuso = this.lt_ps.tuso;
  this.th_ps.mauso = this.lt_ps.mauso;
}
```

Cài đặt phương thức ghi

```
public void Ghi(string strFilename)
{
  this.lt_ps.tuso = this.th_ps.tuso;
  this.lt_ps.mauso = this.th_ps.mauso;
  this.lt_ps.GhiPhanSo(strFilename);
}
```

Cài đặt toán tử +

```
public static XL_PHANSO operator +(XL_PHANSO ps1,XL_PHANSO ps2)
{
  XL_PHANSO kq = new XL_PHANSO(new LT_PHANSO(), new
TH_PHANSO());
  kq.th_ps.tuso = ps1.th_ps.tuso * ps2.th_ps.mauso +
ps2.th_ps.tuso * ps1.th_ps.mauso;
  kq.th_ps.mauso = ps1.th_ps.mauso * ps2.th_ps.mauso;
  return kq;
}
```

Cài đặt hàm cập nhật từ đối tượng xử lý phân số khác

```
public void CapNhat(XL_PHANSO ps)
{
  this.th_ps.tuso = ps.th_ps.tuso;
  this.th_ps.mauso = ps.th_ps.mauso;
}
```

Cài đặt hàm rút gọn phân số

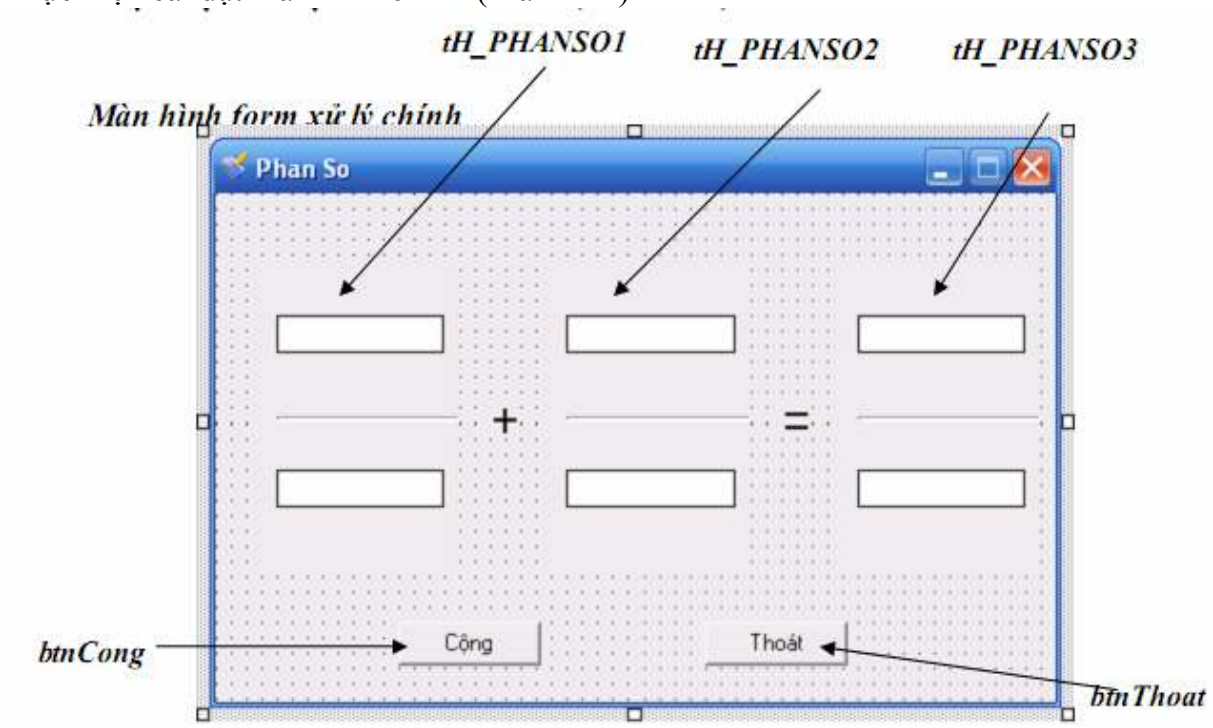
```
public void RutGon()
{
  int tuso = this.th_ps.tuso;
  int mauso = this.th_ps.mauso;
  int maxUC = TimMaxUocChung(tuso,mauso);
  tuso = tuso/maxUC;
  mauso = mauso/maxUC;
```

```
  this.th_ps.tuso = tuso;
  this.th_ps.mauso = mauso;
}
```

Để rút gọn ta cần tính ước chung lớn nhất, có thể cài đặt hàm này chung với lớp XL_PHANSO

```
public int TimMaxUocChung(int a, int b)
{
  while(a!=b)
 {
   if(a>b)
   a -= b;
   else
   b -= a;
 }
  return a;
}
```

Thực hiện cài đặt màn hình chính (MainFrm)



Trong form chính sẽ thực hiện khai báo 3 đối tượng xử lý phân số

```
private PhanSo.XL_PHANSO xl_PhanSo1;
private PhanSo.XL_PHANSO xl_PhanSo2;
private PhanSo.XL_PHANSO xl_PhanSo3;
```

Thực hiện khởi tạo 3 đối tượng xử lý phân số vừa khai báo

```
public MainFrm()
{
  //
  // Required for Windows Form Designer support
 InitializeComponent();
  xl_PhanSo1 = new XL_PHANSO(new LT_PHANSO("phanso1.xml"),
                             tH_PHANSO1);
```

```
 xl_PhanSo2 = new XL_PHANSO(new LT_PHANSO("phanso2.xml"),
                          tH_PHANSO2);
 xl_PhanSo3 = new XL_PHANSO(new LT_PHANSO(""),tH_PHANSO3);
}
```

Viết hàm xử lý cho các nút chức năng trên form:

Hàm xử lý cho nút Cộng

```
private void btnCong_Click(object sender, System.EventArgs e)
{
  XL_PHANSO kq = xl_PhanSo1 + xl_PhanSo2;
 xl_PhanSo3.CapNhat(kq);
 xl_PhanSo3.Ghi("ketqua.xml");
 xl_PhanSo3.RutGon();
}
```

Hàm xử lý cho nút Thoat

```
 private void btnThoat_Click(object sender, System.EventArgs e)
{
  this.Close();
}
```

Tạo các tập tin phanso1.xml, phanso2.xml, có định dạng như ví dụ ở trên.

Thực hiện biên dịch và chạy thử chương trình.

Nhận xét :

Thực hiện cài đặt với kiến trúc 3 lớp sẽ giúp chương trình dễ dàng thay đổi, tái sử dụng lại chương trình.

Ví dụ:

TH_PHANSO không thể hiện tử số và mẫu số bằng TextBox nữa mà thay bằng control khác (ví dụ như MyControl thì cũng không ảnh hưởng, lúc đó chỉ cần thay đổi code trong phần property tử số và mẫu số mà thôi.

```
public int tuso{
  set{
   this.MyControl.Value = value.ToString();
 }
  get{
   return int.Parse(this.MyControl.Value);
 }
}
public int mauso {
  set  {
   this.MyControl.Value = value.ToString();
 }
  get  {
   return int.Parse(this.MyControl.Value);
 }
}
```

Khi không lưu trữ bằng XML mà chuyển sang dùng cơ sở dữ liệu thì ta chỉ cần thay code phần LT_PHANSO, mà không cần thay đổi code phần TH_PHANSO, cũng như XL_PHANSO.

Chú ý:

Không phụ thuộc phương pháp lập trình.

Mỗi nghiệp vụ không nhất thiết chỉ được giải quyết bởi 3 đối tượng.

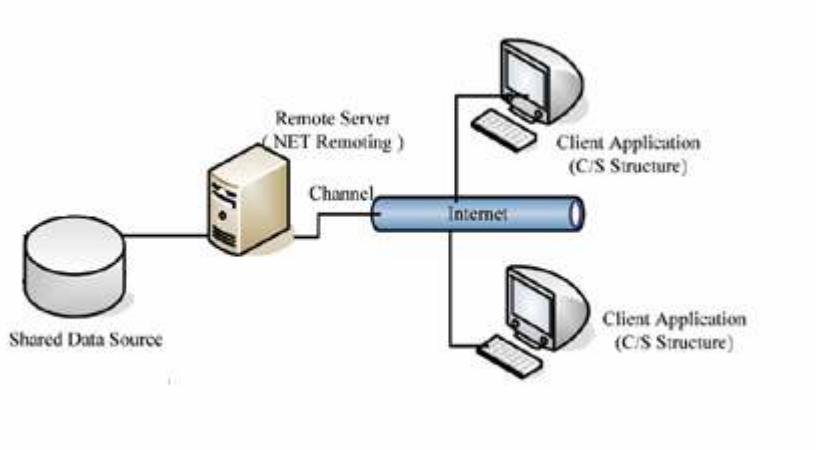
Không là một kiến trúc "siêu việt".

4.2. Remoting

4.2.1. Giới thiệu về Remoting

.NET Remoting là gì?

- Trước hết .NET Remoting là một kĩ thuật .NET được giới thiệu từ .NET framework 1.1. Cùng với .NET Webservice, .NET remoting là lựa chọn cho giải pháp xử lý tính toán từ xa. .NET Remoting là một kĩ thuật cho phép một đối tượng này truy xuất đến một đối tượng khác nằm ở các Application Domain khác nhau. Và nếu giải thích theo kiểu bình dân, ta có thể sử dụng .NET Remoting đế gọi một chương trình hoặc một service chạy trên một máy vi tính khác để xử lý một cái gì đó và trả kết quả tính toán lại cho ta.



Hình 4.1: *.NET Remoting Overview*

.NET Remoting và Distributed COM

- Vào năm một ngàn chín trăm hồi đó, người ta thường thực hiện việc giao tiếp giữa các process bằng cách sử dụng Distributed COM hay còn gọi là DCOM. DCOM đã rất hữu ích cho những chương trình chạy trên các máy tính cùng loại và nằm trong cùng một mạng. Tuy nhiên, DCOM trở nên lỗi thời vì nó không thể chạy trên Internet. DCOM dựa trên một tập giao thức mà không phải object nào cũng hỗ trợ và điều này khiến DCOM không chạy được trên những platform khác nhau. Ngoài ra, DCOM sử dụng nhiều port trong khi các port ấy thường bị chặn bởi firewall. Tất nhiên mở những port đó để nó hoạt động được không khó nhưng đó là một trong những phiền phức.
- .NET Remoting khắc phục những yếu kém của DCOM bằng cách hỗ trợ nhiều giao thức khác nhau.

.NET Remoting và Web Services

- Về khía cạnh xử lý từ xa thì Web Services hoàn toàn tương tự như .NET Remoting. Thậm chí người ta có thể làm cho .NET Remoting trở thành 1 Web Services bằng cách

host nó trong IIS. Web Services cho phép các ứng dụng có thể giao tiếp với nhau mà không phụ thuộc platform, ngôn ngữ lập trình, … Tuy nhiên Web Services là một môi trường "stateless", có nghĩa là nó không lưu lại bất kì trạng thái gì của lần gọi trước và nó cũng không biết gì về phía client đang thực hiện request. Client và server Web Services chỉ có thể trao đổi với nhau bằng các thông điệp SOAP. Những điều sau đây là các điểm khác nhau chính giữa .NET Remoting và Web Serices, chúng cũng là những nhân tố để ta chọn lựa giữa 2 công nghệ này:

• ASP.NET Web Services chỉ có thể được truy xuất qua HTTP còn .NET Remoting có thể được dùng trên nhiều giao thức khác nhau như TCP, HTTP.

• Web Services là một môi trường stateless. Khi có một request từ phía client, sẽ có một object mới được tạo ra để thực hiện request đó trên server. Còn .NET Remoting lại hỗ trợ nhiều lựa chọn state management và có thể thực hiện nhiều request từ một client, đồng thời có hỗ trợ callbacks.

• Web Services serialize các đối tượng thành XML bên trong SOAP message và vì thế có thể truyền tải thông tin của bất cứ thành phần nào miễn có thể chuyển thành XML. Còn đối với .NET Remoting thì tùy giao thức và định dạng message mà nó có thể truyền đi thông tin như thế nào. Ngoài ra theo như giới thiệu thì .NET Remoting có cho phép đối tượng được truyền vào theo cả kiểu tham chiếu(reference) và tham trị (value)

• Web services có thể hoạt động trên các platform môi trường khác nhau trong khi .NET Remoting yêu cầu phía clients phải là .NET application.

Channels

- Trong kĩ thuật .NET Remoting thì Channel được hiểu như là một kênh để giao tiếp giữa client và server. Một object từ client sẽ thông qua Channel để giao tiếp với object phía server, Channel sẽ truyền tải những message từ hai phía. Như giới thiệu phía trên thì có hai channel chính là TcpChannel và HttpChannel tương ứng với các giao thức TCP và HTTP. Ngoài ra, TcpChannel và HttpChannel đều có khả năng extend thành những Custom Channel của bạn.

Làm sao để tạo một Object có thể Remote được trong .NET Remoting?

- Một Object remote được chỉ là một object thông thường nhưng phải được inherit từ MarshalByRefObject. Đoạn code sample ở hình 4.2 là một ví dụ đơn giản về Remotable Object. Đối tượng SampleObject trong hình có một số method đơn giản trả về phép tính tổng, hiệu, tích, thương của hai số nguyên. Giá trị trả về của hàm là kiểu số nguyên, kiểu built-in của .NET framework. Nếu bạn muốn trả về kiểu dữ liệu bạn tự định nghĩa, hoặc một instance của class bạn định nghĩa thì lớp đó của bạn phải được khai báo với attribute Serializable.

```
using System;
public class SampleObject: MarshalByRefObject
{
  public int Add(int a, int b)
  {
   int c = a + b;
```

```
    return c;
  }
  public int Subtract(int a, int b)
  {
    int c = a - b;
    return c;
  }
  public int Multiply(int a, int b)
  {
    int c = a * b;
    return c;
  }
  public int Divide(int a, int b)
  {
    int c;
    if (b != 0)
      c = a / b;
    else
      c = 0;
    return c;
  }
}
```

Hình 4.2: *Remotable Object Sample*

Tạo chương trình Server để host Remotable Object

- Kế tiếp, chúng ta cần tạo ra một chương trình server để lắng nghe những request từ phía client. Trong ví dụ này chúng ta sẽ sử dụng TCP/IP channel. Đầu tiên chúng ta tạo một instance channel và đăng kí một port tương ứng cho nó. Khi có một Request từ phía client, server sẽ nhận request đó và Remote Object của chúng ta sẽ thực thi Request này. Trong .NET Remoting, có hai cơ chế để tạo instance của Remote Object rồi từ đó thực thi request: Singleton và Singlecall. Tùy vào mục đích sử dụng, nhu cầu của chương trình mà server của bạn có thể khai báo theo cơ chế WellKnownObjectMode.SingleCall, hay WellKnownObjectMode.Singleton. Khi khai báo Singleton, Remote Object sẽ được sinh ra, thực thi request, reply lại phía client và sau đó, object này vẫn được lưu lại chứ không bị hủy đi. Đến khi nào process chạy chương trình server kết thúc thì instance này mới bị trình hốt rác Garbage Collector hốt đi. Và ngược lại, khi khai báo là SingleCall, Remote Object sẽ được khởi tạo và hủy đi đối với mỗi lần nhận request từ phía client, cơ chế này tương tự như mô hình .NET Web Service truyền thống.
- Nếu bạn muốn sử dụng .NET Remoting trong IIS thì không cần tạo một chương trình server như thế này. Và tất nhiên, IIS chỉ hỗ trợ HttpChannel. Nếu host 1 .NET Remoting bên trong IIS bạn sẽ mặc nhiên sử dụng được cơ chế Authentication của IIS, ngược lại nếu làm một chương trình server để host như trên thì bạn phải cài đặt cơ chế Authentication của riêng mình. Để host một Remote Object bên trong IIS, trước tiên phải tạo 1 Virtual Directory cho application, sau đó đặt đoạn code đăng kí service bên trong event Application_Start (file global.asax)

- Trong ví dụ này, chúng ta sẽ không sử dụng IIS mà sẽ tạo một console application. Có nhiều lựa chọn khi không sử dụng IIS, ta có thể sử dụng console application, Win form application nhưng trong thực tế, người ta sẽ sử dụng một Windows Service để làm. Còn Console application hay Winform Application thường chỉ dùng để minh họa. Trong ví dụ này, chúng ta sẽ sử dụng port 9999 cho may mắn. Có thể một chương trình nào đó trong máy của bạn đã sử dụng port này, nếu bị như vậy bạn phải chọn port khác. Và sau cùng, để kiểm tra xem máy bạn đang lắng nghe trên những port nào (port nào đã bị sử dụng) thì ta dùng lệnh "netstat –a" trong command prompt.
- Còn bây giờ, hãy xem một console application project với 1 class tên là SampleSerrver. Trong project này tôi đã thêm reference tới System.Runtime.Remoting vào trong project để nó có thể chạy được.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
public class  Server
{
 public static int Main()
 {
  TcpChannel chan = new TcpChannel(9999);
  ChannelServices.RegisterChannel(chan, false);
  RemotingConfiguration.RegisterWellKnownServiceType(typeof(SampleObject)
        , "SampleNetRemoting", WellKnownObjectMode.SingleCall);
  Console.WriteLine("Hit <enter> to exit...");
  Console.ReadLine();
  }
}
```

Hình 4. 3: *Sample Server host Remotable Object*

Tạo chương trình client để sử dụng Remote Object.

- Chương trình client trong ví dụ này cũng khá đơn giản, nó sẽ connect vào server, tạo một instance của Remote Object và excute method tính tổng, hiệu, tích, thương.
- Các bạn lưu ý rằng trong cả chương trình client và chương trình server đều phải reference tới class SampleObject. Client sẽ gọi method của instance SampleObject, nhưng server sẽ thực thi xử lý nó chứ không phải phía client.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
public class Client
{
```

```
  public static int Main (string[] argv)
  {
   TcpChannel chan = new TcpChannel();
   ChannelServices.RegisterChannel(chan, false);
   SampleObject obj = (SampleObject)Activator.GetObject(
     typeof(SampleObject), "tcp://localhost:9999/SampleNetRemoting");
   if (obj == null)
     System.Console.WriteLine("Could not locate server");
   else
   {
     int a = Convert.ToInt32(argv[0]);
     int b = Convert.ToInt32(argv[1]);
     int c = obj.Add(a, b);
     Console.WriteLine("a + b = {0}", c);
     c = obj.Subtract(a, b);
     Console.WriteLine("a - b = {0}", c);
     c = obj.Multiply(a, b);
     Console.WriteLine("a * b = {0}", c);
     c = obj.Divide(a, b);
     Console.WriteLine("a / b = {0}", c);
   }
     Console.ReadKey();
  }
}
```

Hình 4: *Sample Client Application*

Test thử chương trình

- Trước tiên chạy chương trình server, bạn sẽ thấy message "Press the enter key to exit" trong cửa sổ console. Như vậy server của bạn đang lắng nghe trên port 9999. Bây giờ bạn hãy chạy chương trình client và sẽ nhìn thấy kết quả trả về trên màn hình. Bạn có thể chạy nhiều client để cùng request đến 1 server nhưng không thể chạy nhiều server. Bạn có thể copy chương trình server sang một máy của bạn mình và nhờ chạy thử, còn bạn sửa lại chương trình client, sửa "localhost" thành IP của máy bạn mình và chạy thử để thấy kết quả.

Tóm tắt:

- Ví dụ ở trên đã sử dụng code C# để khai báo các cấu hình cho server và client tuy nhiên .NET Remoting cho phép ta cấu hình trước trong file config (App.config). Các bạn có thể tham khảo một số resource phía dưới để biết cách làm.
- .NET Remoting là một trong những kĩ thuật tiện lợi cho những chương trình dạng Distributed Computing. Cách sử dụng nó phức tạp hơn Web Service tuy nhiên nếu bạn

muốn tăng performance thì .NET Remoting với Singleton và TCP channel sẽ là lựa chọn rất tốt.

- Với sự ra đời của .NET Framework 3.x, Microsoft đã giới thiệu nền tảng mới hơn cho các kĩ thuật RPC, đó là WCF mạnh hơn .NET Remoting rất nhiều.

4.2.2. Khai báo, cài đặt và đăng ký giao diện từ xa

Để cho chương trình có tính khả chuyển cao thay vì người ta xây dựng lớp Remote Object như ví dụ trên chúng ta khai báo một giao diện là lớp Remote Object và trong chương trình phía Server ta sẽ cài đặt giao diện này và đăng ký giao diện từ xa. Như vậy để triển khai một hệ thống Remoting ta có 3 chương trình: Giao diện Remote Object, chương trình Server triển khai giao diện và đăng ký giao diện từ xa, chương trình Client triệu gọi phương thức từ xa.

- Khai báo giao diện từ xa
- Cài đặt và đăng ký giao diện từ xa

4.2.3. Triệu gọi phương thức từ xa

- Chương trình phía Client chúng ta triệu gọi phương thức được cung cấp bởi giao diện tử xa đã được đăng ký và cung cấp bởi Server

4.3. Web Services

4.3.1. Giới thiệu về Web Services

1. Web Service là gì?

Web service là một Modul chương trình cung cấp chức năng của các ứng dụng cho phép triệu gọi và truy cập từ xa thông qua Internet. Web service sử dụng các chuẩn của Internet như XML và HTTP. Việc sử dụng Web service phụ thuộc nhiều vào sự chấp nhận của XML, một ngôn ngữ mô tả dữ liệu mới dùng để truyền tải dữ liệu thông qua Web.

Bất kỳ một Web service nào cũng có thể được sử dụng, hoặc là trong ứng dụng cục bộ hoặc truy cập từ xa qua Internet bởi nhiều ứng dụng. Do có khả năng truy cập qua các giao diện chuẩn mà một Web service cho phép nhiều hệ thống khác nhau cùng làm việc với nhau như một tiến trình duy nhất trên Web.

2. Vai trò của Web service

Web service ra đời đã mở ra một hướng mới cho việc phát triển các ứng dụng trên Internet. Web services tạm dịch là các dịch vụ trên web. Công nghệ web services ra đời là một cuộc cách mạng hóa cách thức hoạt động của các dịch vụ B2B và B2C. Web services kết hợp sử dụng nhiều công nghệ khác nhau cho phép hai ứng dụng cùng ngôn ngữ, độc lập hệ điều hành trao đổi được với nhau thông qua môi trường mạng Internet. Tuy nhiên những công nghệ sử dụng ở đây không nhất thiết phải là những công nghệ mới. Đây là điểm khác biệt của web services so với các công nghệ khác, đó chính là khả năng kết hợp các công nghệ đã có như là XML, SOAP, WSDL, UDDI để tạo ra các service, đặc điểm này làm nổi bật vai trò của web services.

Web Service được thiết kế nhằm cung cấp một cơ chế cho phép các chương trình giao tiếp với nhau qua Internet (sử dụng các giao thức Internet như HTTP GET, HTP POST và SOAP).

3. Đặc điểm Web service

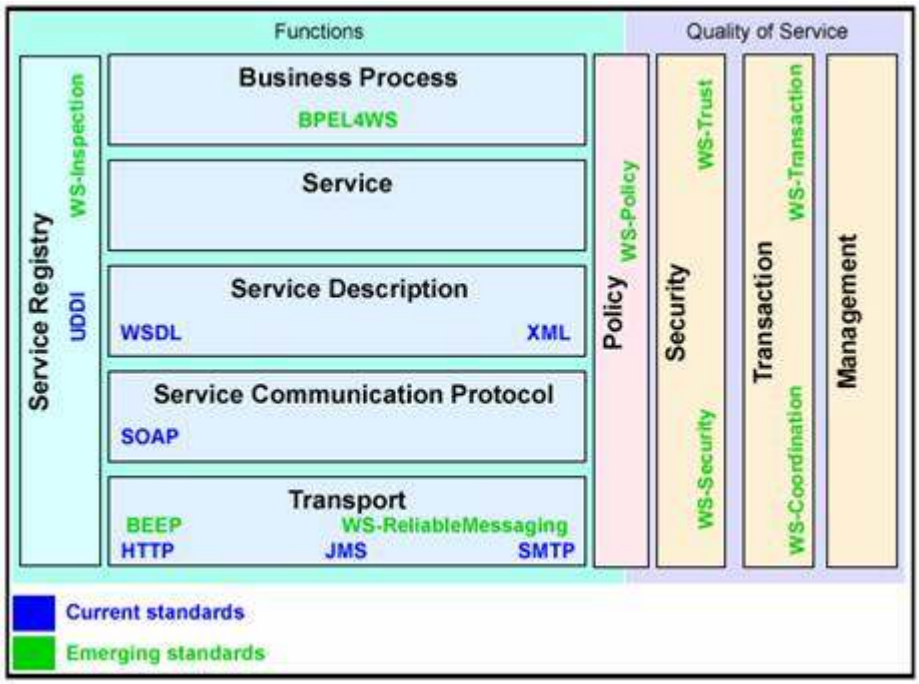
- Web service cho phép client và server tương tác được với nhau mặc dù trong những môi trường khác nhau.
- Web Service thì có dạng mở và dựa vào các tiêu chuẩn XML và HTTP là nền tảng kỹ thuật cho web service. Phần lớn kỹ thuật của web service được xây dựng là những dự án nguồn mở. Bởi vậy chúng độc lập và vận hành được với nhau.
- Web Service thì rất linh động: Vì với UDDI và WSDL, thì việc mô tả và phát triển web service có thể được tự động hoá.
- Web service được xây dựng trên nền tảng những công nghệ đã được chấp nhận.
- Web service có dạng modul.
- Web service có thể được công bố (publish) và gọi thực hiện qua mạng.

Ngày nay web service được sử dụng rất nhiều trong những lĩnh vực khác nhau của cuộc sống như:

- Dịch vụ chọn lọc và phân loại tin tức: là những hệ thống thư viện kết nối đến các web portal để tìm kiếm các thông tin từ các nhà xuất bản có chứa những khoá muốn tìm.
- Dịch vụ hiển thị danh sách đĩa nhạc dành cho các công ty thu thanh.
- Ứng dụng đại lý du lịch có nhiều giá vé đi du lịch khác nhau do có chọn lựa phục vụ của nhiều hãng hàng không.
- Bảng tính toán chính sách bảo hiểm dùng công nghệ Excel/COM với giao diện web.
- Thông tin thương mại bao gồm nhiều nội dung, nhiều mục tin như: dự báo thời tiết, thông tin sức khoẻ, lịch bay, tỷ giá cổ phiếu,…
- Những giao dịch trục tuyến cho cả B2B và B2C như: đặt vé máy bay, làm giao kèo thuê xe.
- Hệ thống thông tin dùng java để tính toán tỷ giá chuyển đổi giữa các loại tiền tệ. Hệ thống này sẽ được các ứng dụng khác dùng như một web service.

4. Kiến trúc Web service

Kiến trúc của Web service bao gồm các tầng như sau:



Hình 1: Kiến trúc Web service

Trong đó bao gồm các tầng như sau:

- Tầng vận chuyển: có nhiệm vụ truyền thông điệp giữa các ứng dụng mạng, bao gồm những giao thức như HTTP, SMTP, FTP, JSM và gần đây nhất là giao thức thay đổi khối mở rộng (Blocks Extensible Exchange Protocol- BEEP).
- Tầng giao thức tương tác dịch vụ (Service Communication Protocol) với công nghệ chuẩn là SOAP. SOAP là giao thức nằm giữa tầng vận chuyển và tầng mô tả thông tin về dịch vụ, SOAP cho phép người dùng triệu gọi một service từ xa thông qua một message XML.
- Tầng mô tả dịch vụ (Service Description) với công nghệ chuẩn là WSDL và XML. WSDL là một ngôn ngữ mô tả giao tiếp và thực thi dựa trên XML. Web service sử dụng ngôn ngữ WSDL để truyền các tham số và các loại dữ liệu cho các thao tác, các chức năng mà web service cung cấp.
- Tầng dịch vụ (Service): cung cấp các chức năng của service.
- Tầng đăng ký dịch vụ (Service Registry) với công nghệ chuẩn là UDDI. UDDI dùng cho cả người dùng và SOAP server, nó cho phép đăng ký dịch vụ để người dùng có thể gọi thực hiện service từ xa qua mạng, hay nói cách khác một service cần phải được đăng ký để cho phép các client có thể gọi thực hiện
- Bên cạnh đó để cho các service có tính an toàn, toàn vẹn và bảo mật thông tin trong kiến trúc web service chúng ta có thêm các tầng Policy, Security, Transaction, Management giúp tăng cường tính bảo mật, an toàn và toàn vẹn thông tin khi sử dụng service.

4.3.2. Giao thức SOAP

SOAP là chữ viết tắt của cụm từ "Simple Object Access Protocol – Giao thức truy cập đối tượng đơn giản", nhưng với sự xem xét mới nhất thì, SOAP sẽ không còn là một từ viết tắt nữa. Chuẩn SOAP ghi nhận XML được thể hiện thế nào bên trong tài liệu SOAP, làm thế nào nội dung của thông điệp được truyền tải, và làm thế nào thông điệp được xử lý ở cả hai phía gởi và nhận. SOAP cũng cung cấp một tập các từ vựng chuẩn.

Các thuật ngữ:

Như bất kỳ công nghệ nào, SOAP cũng có tập các thuật ngữ của riêng nó. Có nhiều thuật ngữ được sử dụng thường xuyên để mô tả các khía cạnh khác nhau của chuẩn SOAP. Nhiều lập trình viên dùng các thuật ngữ này mà không thật sự hiểu ý nghĩa của nó. Để hiểu thật sự các khái niệm đòi hỏi phải tốn một thời gian để hiểu ý nghĩa của từng thuật ngữ và làm thế nào để áp dụng cho cả chuẩn SOAP và một Web Services thực thụ

Chú ý: Chuẩn SOAP không chỉ là chuẩn XML mà chuẩn này còn bao gồm các thông điệp SOAP có hành vi như thế nào, các phương tiện vận chuyển khác nhau, cách mà các lỗi được xử lý..

Sự truyền tải dữ liệu

SOAP Binding

Thuật ngữ mô tả làm thế nào một thông điệp SOAP tương tác được với một giao thức vận chuyển như HTTP, SMTP hay FTP để di chuyển trên Internet. Điều quan trọng là SOAP di chuyển bằng một giao thức chuẩn để liên lạc với các sản phẩm Web Services khác.

Trước SOAP, nhiều người phát triển đã tạo ra các phương pháp của riêng họ để chuyển tải một tài liệu XML trên mạng. Các cách này vẫn hoạt động tốt trong phạm vi một nhóm cụ thể. Tuy nhiên khi bạn cần làm việc với một nhóm khác ở trong hay bên ngoài công ty thì điều này trở nên khó khăn vì phải huấn luyện và có thể thay đổi để làm việc với việc truyền tải tài liệu XML mà họ đang sử dụng. Bằng cách sử dụng một tài liệu XML chuẩn trên các giao thức chuẩn, công việc cần làm khi cộng tác với nhau sẽ được giảm thiểu tối đa

SOAP Message Exchang Pattern (MEP)

Thuật ngữ mô tả làm thế nào mà một tài liệu SOAP trao đổi giữa phía máy khách và chủ. Thông điệp SOAP sở hữu một liên kết, như là HTTP, để nó có thể truyền trên Internet. Việc nói chuyện giữa máy khách và chủ, ở đây là các nút, xác định các hành động mà cả hai phía thực hiện.

Nhắc lại SOAP là một XML đóng gói RPC. Vì thế, MEP hoàn toàn là yêu cầu và phản hồi giữa máy khách và chủ (hay các nút khác). Như vậy nếu có nhu cầu liên lạc giữa các nút thì việc này được thực hiện bằng nhiều yêu cầu và phản hồi để hoàn tất việc truyền thông điệp. Cách này khác hẳn với các công nghệ đối tượng từ xa khác như CORBA, công nghệ đó được thực hiện chỉ trong một kết nối.

SOAP Application

Một ứng dụng SOAP đơn giản là một ứng dụng dùng SOAP theo một vài cách khác nhau. Vài ứng dụng hoàn toàn dựa trên chuẩn SOAP, như là Web Services cổ phiếu, hoặc dùng chuẩn SOAP để nhận mã và các cập nhật của phần mềm. Chú ý là một ứng dụng có thể tạo, sử dụng hoặc là nút trung gian của Web Services.

SOAP Node

Trách nhiệm của một nút có thể bao gồm gởi, nhận, xử lý hoặc truyền tải lại một thông điệp SOAP. Một nút chỉ là một phần nhỏ của phần mềm, xử lý một tài liệu SOAP phụ thuộc vào

vai trò của nó. Bên cạnh việc truyền dữ liệu, một nút có trách nhiệm đảm bảo thông tin XML trong tài liệu SOAP phải đúng ngữ pháp theo chuẩn SOAP.

SOAP Role

Một vai trò của SOAP định nghĩa một nút cụ thể hoạt động như thế nào. Nó có thể là nút gởi, nhận hoặc nút trung gian.

SOAP Sender

Nút gởi là nút gởi yêu cầu SOAP. Nếu bạn nghĩ đến ví dụ của ứng dụng khách chủ thì khi ứng dụng khách thực hiện yêu cầu, nó gởi thông điệp tới ứng dụng chủ để yêu cầu vài thông tin.

SOAP Receiver

Ngược lại với SOAP sender là nút nhận.

SOAP Intermediary

Một nút trung gian có thể xem một thông điệp SOAP và tương tác trên vài phần thông tin của thông điệp, và chuyển đến vị trí kế tiếp của thông điệp. Một nút trung gian thường hoạt động như một router. Một router sẽ xem xét thông tin của gói tin chuyển trên mạng, tìm điểm kế tiếp của gói tin và chuyển gói tin đến đó.

Message Path

Một thông điệp SOAP di chuyển từ phía bên gởi đến phía bên nhận thông điệp thông qua nhiều nút trung gian. Tuyến đường đi của thông điệp được gọi là một Message Path.

Initial SOAP Sender

Nút gởi yêu cầu SOAP đầu tiên là nút gởi SOAP ban đầu.

SOAP Feature

Một đặc điểm SOAP là một phần chức năng của phần mềm hỗ trợ chức năng SOAP.

Các thuật ngữ liên quan đến XML

Chuẩn SOAP cũng định nghĩa một tập nhỏ các phần tử XML để đóng gói dữ liệu được truyền giữa các nút. Thật sự chỉ có vài phần tử vì phần thân của thông điệp có thể khác nhau phụ thuộc vào cài đặt. Sự uyển chuyển này được cho phép bởi chuẩn SOAP.

SOAP Message

Đây là tài liệu XML được truyền bởi một nút SOAP gởi hoặc nhận. Một nút gởi hoặc nút khách tạo ra một tài liệu XML chứa thông tin mà phía bên khách cần từ phía chủ. Một khi tài liệu được truyền, phía bên chủ phân giải thông tin trong tài liệu để truy xuất các giá trị khác nhau và tạo một thông điệp SOAP mới để phản hồi.

SOAP Envelope

Đây là phần tử gốc của tài liệu SOAP XML. Tài liệu SOAP chứa nhiều định nghĩa không gian tên (namespace) nhưng các phần tử liên quan tới thông điệp SOAP sẽ có ENV: là tiếp đầu ngữ.

SOAP Header

Phần đầu của một thông điệp SOAP chứa một khối thông tin đầu trong tài liệu XML để định tuyến và xử lý thông điệp SOAP. Dữ liệu này tách rời khỏi phần thân của tài liệu có chứa thông tin liên quan đến đối tượng được gọi.

SOAP Header Block

Phần đầu của SOAP chứa nhiều phần giới hạn hay là nhiều khối thông tin có một khối thông tin của phần đầu. Những khối thông tin của phần đầu này chứa thông tin về các nút trung gian vì một nút cần biết nút kế tiếp để thông điệp được gởi đến.

SOAP Body

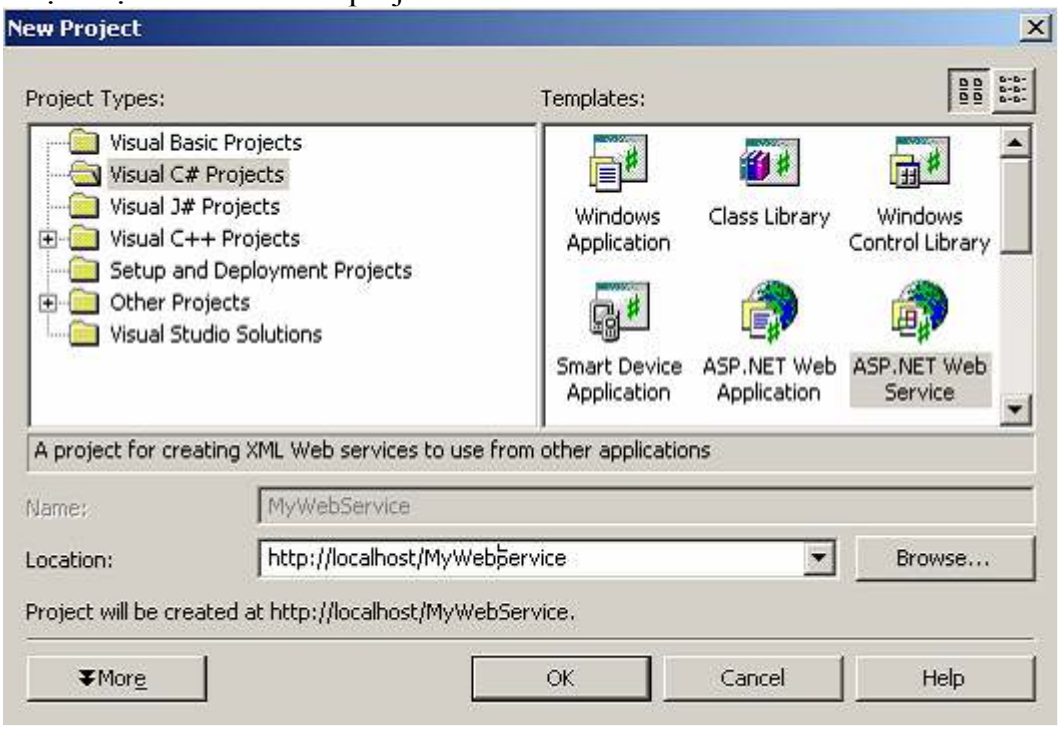
Phần thân của SOAP thật sự chứa thông tin của đối tượng để xử lý thông tin. Phần thân sau khi được phân tách sẽ trở thành đối tượng. Đối tượng xử lý thông tin và kết quả được đặt trong phần thân của tài liệu trả về.

SOAP Fault

Đây là một phần thông tin của SOAP chứa thông tin đến bất kỳ lỗi gì xảy ra tại một nút SOAP.

4.3.3. Xây dựng Web Services

- Tạo một Web Services project

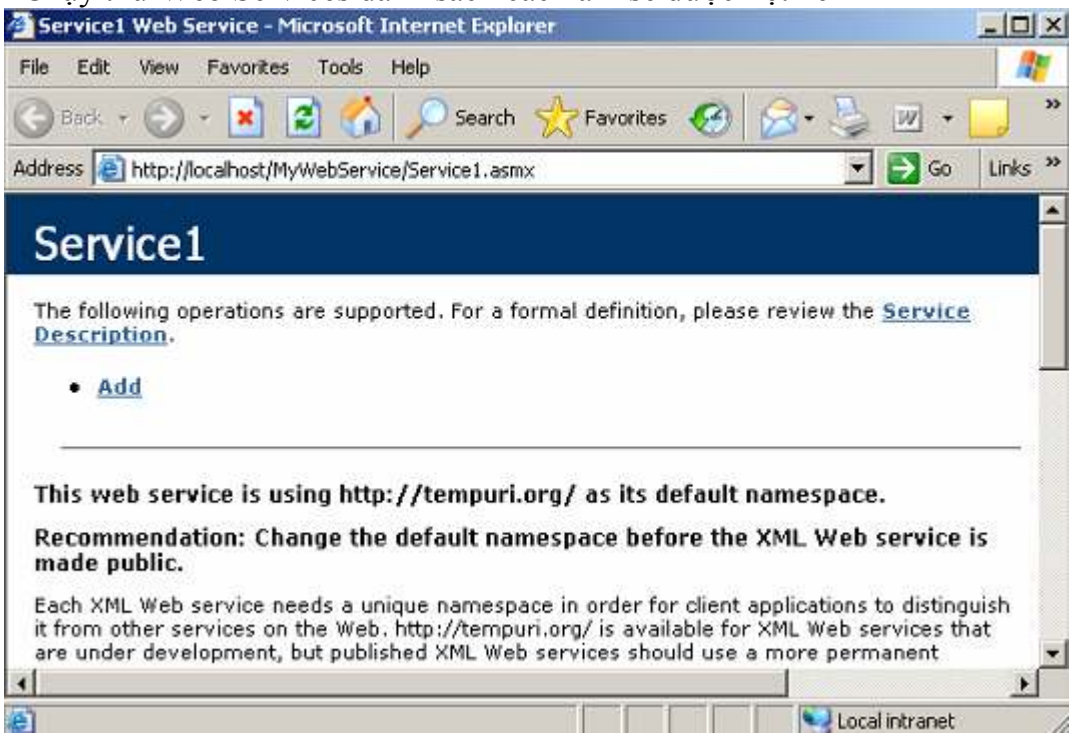


- Tạo Web Method

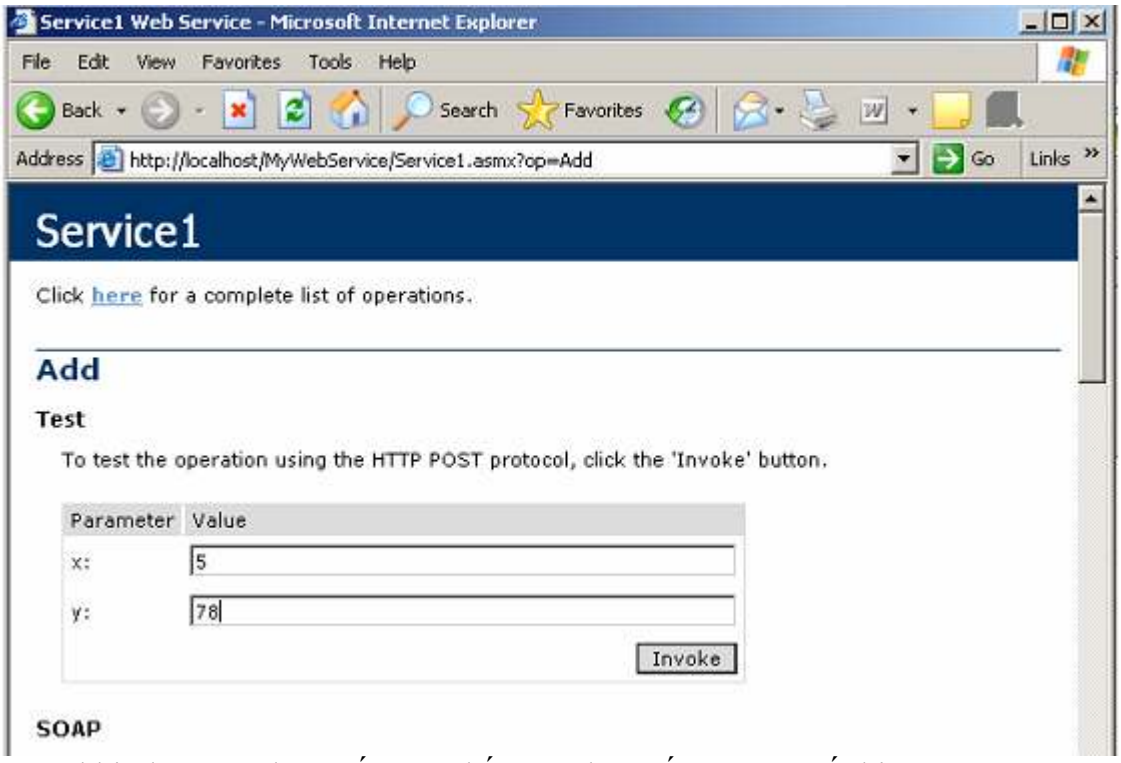
```
MyWebService.Service1                                   Service1()
namespace MyWebService
{
    /// <summary>
    /// Summary description for Service1.
    /// </summary>
    public class Service1 : System.Web.Services.WebService
    {
        public Service1()
        {
            //CODEGEN: This call is required by the ASP.NET Web Services Designer
            InitializeComponent();
        }
        Component Designer generated code

        [WebMethod]
        public int Add(int x, int y)
        {
            return x+y;
        }
    }
}
```

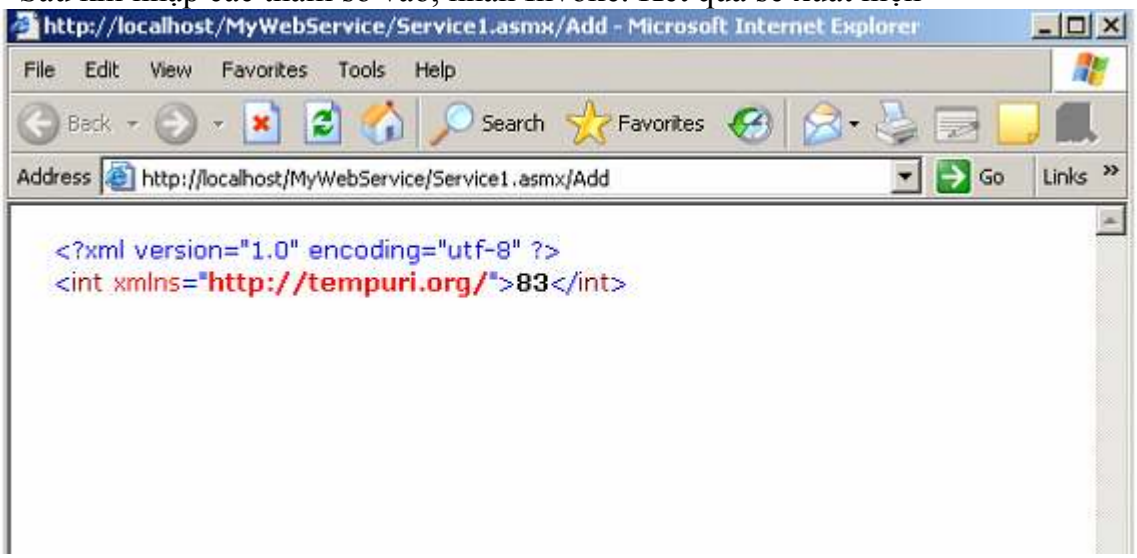
- Chạy thử Web Services danh sách các hàm sẽ được liệt kê



- Chọn hàm Add



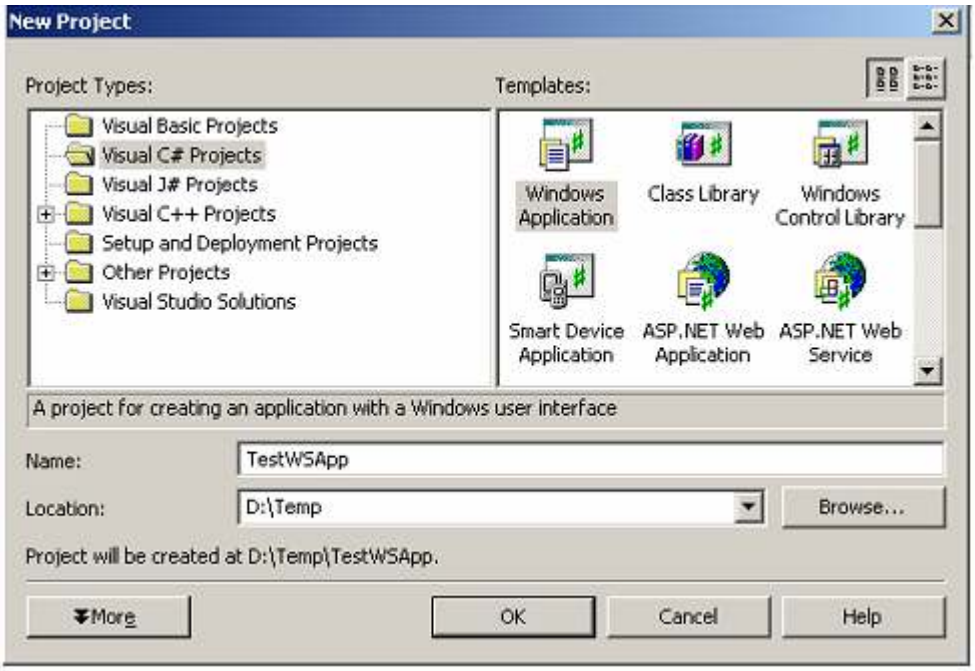
- Sau khi nhập các tham số vào, nhấn Invoke. Kết quả sẽ xuất hiện



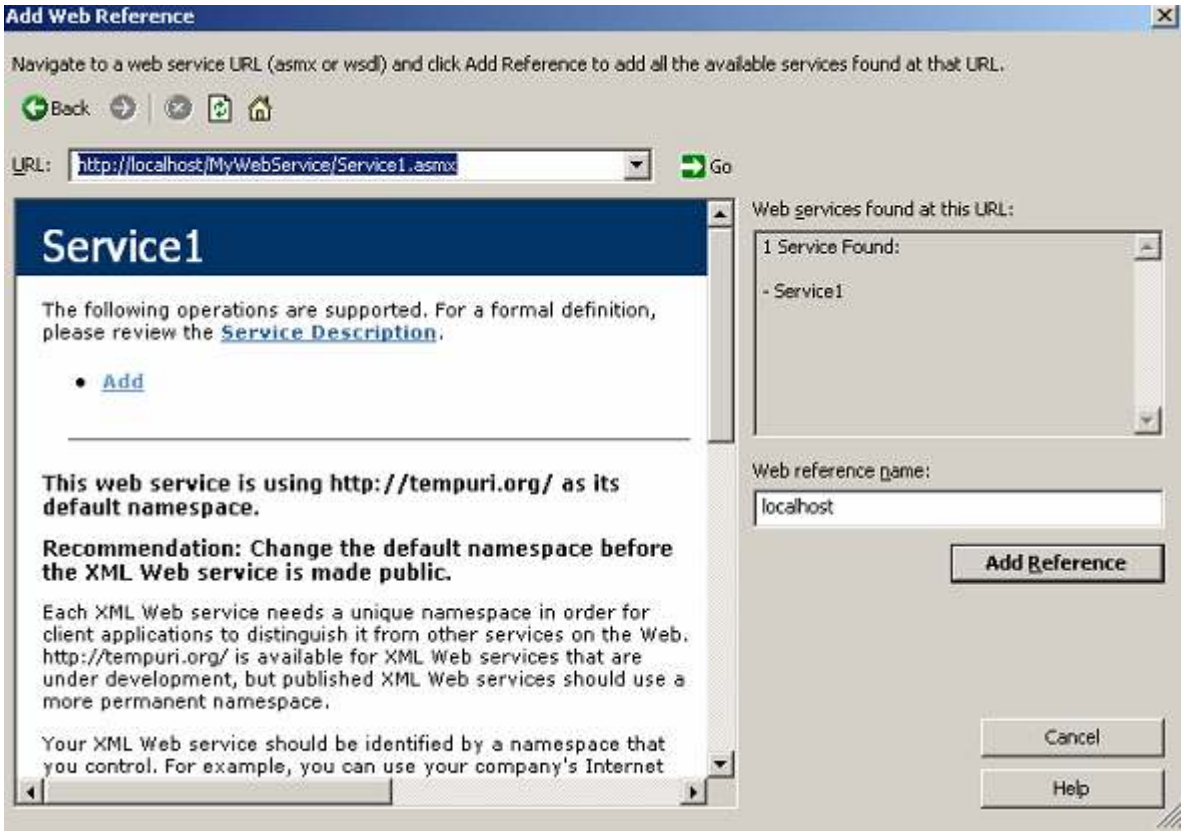
4.3.4. Triệu gọi Web Services từ ứng dụng .NET, Java và các ngôn ngữ khác

- Sau khi xây dựng Web Server song ta có thể triệu gọi nó từ một ứng dụng khác

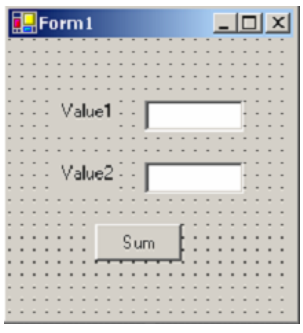
- Tạo một Window Form



- Add Web Reference



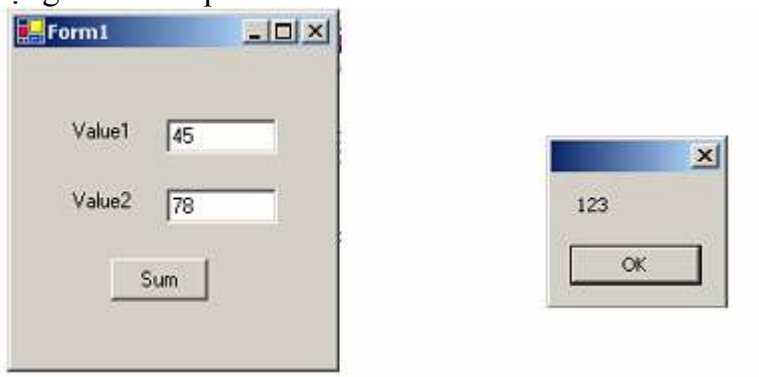
- Tạo màn hình:



- Viết hàm xử lý nút nhấn:

```
private void btnSum_Click(object sender, System.EventArgs e)
{
    localhost.Service1 ws = new localhost.Service1();
    int sum;
    sum = ws.Add(Convert.ToInt32(txtValue1.Text), Convert.ToInt32(txtValue2.Text));
    MessageBox.Show(sum.ToString());
}
```

- Chạy thử ứng dụng ta có kết quả:



4.4 Thảo luận về các ứng dụng phân tán

4.5. Bài tập áp dụng

1. Viết chương trình Chat sử dụng công nghệ Web Services
2. Viết chương trình Calculator bằng công nghệ Web Services
3. Viết chường trình quản lý FileManager bằng công nghệ Web Services.

TÀI LIỆU THAM KHẢO

1. Richard Blum, C# Network Programming, 2003
2. Fiach Reid, Network programming in NET with C# and VB.NET, Digital Press, 2003
3. Bài giảng "Nhập môn Công nghệ phần mềm", Đại học KHTN
4. Bài giảng "Xây dựng phần mềm hướng đối tượng", Đại học KHTN
5. Bài giảng "Lập trình truyền thông", Đại học Cần Thơ
6. Bài giảng "Công nghệ .NET", Khoa CNTT – Đại học SPKT Hưng Yên
7. Bài giảng "Java Nâng cao", Khoa CNTT- Đại học SPKT Hưng Yên
8. Các ví dụ tại Website: www.java2s.com