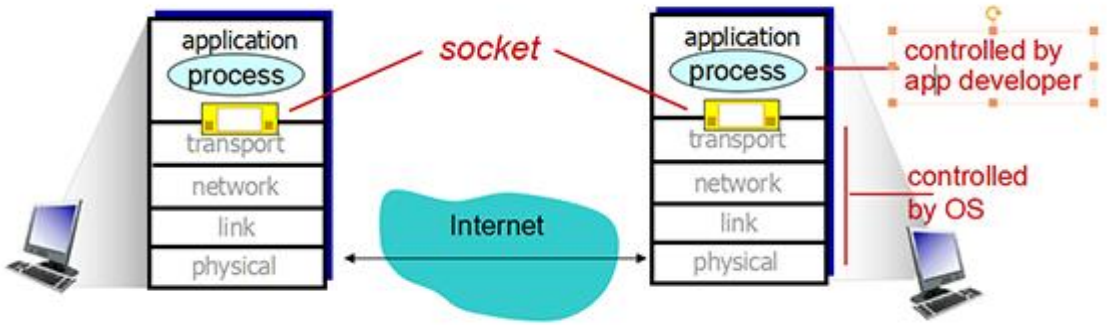
Contents

1) Socket là gì? Cho ví dụ?

Trong lập trình mạng dùng Socket, chúng ta không trực tiếp truy cập vào các thiết bị mạng để gởi và nhận dữ liệu. Thay vì vậy, một file mô tả trung gian được tạo ra để điều khiển việc lập trình. Các file mô tả dùng để tham chiếu đến các kết nối mạng được gọi là các Socket. Socket định nghĩa những đặc trưng sau:

- Một kết nối mạng hay một đường ống dẫn để truyền tải dữ liệu
- Một kiểu truyền thông như stream (TCP), datagram (UDP), raw (hỗ trợ trừu tượng hóa socket, dành cho người dùng nâng cao muốn xây dựng một giao thức riêng), WebSocket, Sequenced packet
- Một giao thức như TCP hay UDP

Socket phải được gắn vào một địa chỉ mạng và một port trên hệ thống cục bộ hay ở xa.



Ví dụ: TCP, UDP, RDS, IP, Internet Control Message Protocol (ICMP)

2) So sánh 2 giao thức UDP và TCP?

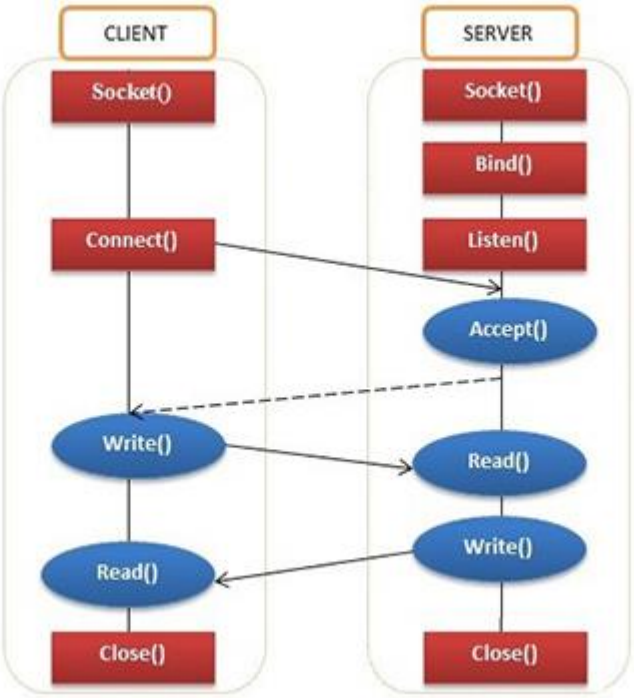
TCP	UDP
Giao thức TCP có độ tin cậy cao, các gói tin được gửi bằng TCP sẽ được theo dõi do vậy dữ liệu sẽ không bị mất hoặc hỏng trong quá trình vận chuyển. Đó là lý do tại sao file tải xuống không bị hỏng ngay cả khi mạng có vấn đề. Giao thức TCP đạt được điều này theo hai cách. Đầu tiên, nó yêu cầu các gói tin bằng cách đánh số chúng. Thứ hai, nó kiểm tra lỗi bằng cách yêu cầu bên nhận	*Không có độ tin cậy cao*. Giao thức UDP hoạt động tương tự như TCP, nhưng nó bỏ qua quá trình kiểm tra lỗi. Khi một ứng dụng sử dụng giao thức UDP, các gói tin được gửi cho bên nhận và bên gửi không phải chờ để đảm bảo bên nhận đã nhận được gói tin, do đó nó lại tiếp tục gửi gói tin tiếp theo. Nếu bên nhận bỏ lỡ một vài gói tin UDP, họ sẽ mất vì bên gửi không

gửi phản hồi đã nhận được cho bên gửi. Nếu bên gửi không nhận được phản hồi đúng, nó có thể gửi lại gói tin để đảm bảo bên nhận nhận chúng một cách chính xác. Các ứng dụng sử dụng giao thức TCP: HTTP, DNS, SMTP, telnet, SNMP v.v.	gửi lại chúng. Do đó thiết bị có thể giao tiếp nhanh hơn. UDP được sử dụng khi tốc độ nhanh và không cần thiết sửa lỗi. Ví dụ, UDP thường được sử dụng cho các chương trình phát sóng trực tiếp, game online, DNS (cả TCP và UDP), VoIP, stream media, nghe nhạc…
TCP là giao thức hướng kết nối (connection-oriented), có nghĩa là buộc phải thiết lập kết nối trước sau đó mới đến tiến trình truyền dữ liệu.	*UDP (User Datagram Protocol) là loại giao thức phi kết nối*, không trạng thái, không cần thiết lập các kết nối trước khi gửi gói tin.
Cơ chế điều khiển luồng trong TCP (Flow Control). Giả sử: Sender gửi quá nhiều dữ liệu cho Receiver, thì Receive sẽ chuyển vào bộ đệm để chờ xử lý, đến lúc bộ đệm đầy thì B gởi tín hiệu cho A để không truyền nữa cho đến khi B xử lý hết thì sẽ gởi lại gói tin cho A để tiếp tục nhận dữ liệu.	*Không có Flow Control*

3) So sánh các loại socket: Stream, Datagram và Raw?

a. Stream Socket

Hay còn gọi là socket hướng kết nối, là socket hoạt động thông qua giao thức TCP (Transmission Control Protocol). Stream Socket chỉ hoạt động khi server và client đã kết nối với nhau.



- Các bước thiết lập một socket phía client gồm:

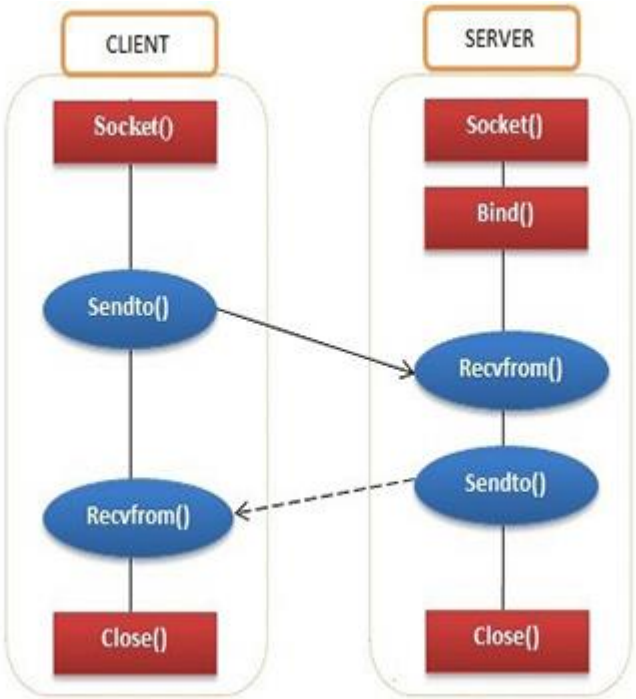
- Tạo một socket bằng hàm socket()
- Kết nối socket đến địa chỉ của server bằng hàm connect()
- Gửi và nhận dữ liệu: Có một số cách khác nhau, đơn giản nhất là sử dụng các hàm read() và write()
- Đóng kết nối bằng hàm close()

- Các bước thiết lập một socket phía server gồm:

- Tạo một socket bằng hàm socket()
- Gắn (bind) socket đến địa chỉ của server sử dụng hàm bind().
- Đối với server trên internet địa chỉ bao gồm địa chỉ ip của máy host + số hiệu cổng dịch vụ (port number)
- Lắng nghe (listen) các kết nối đến từ clients sử dụng hàm listen()
- Chấp nhận các kết nối sử dụng hàm accept(). Hàm này sẽ dừng (block) cho đến khi nhận được một client kết nối đến.
- Gửi và nhận dữ liệu với client (hàm read(), write())
- Đóng kết nối bằng hàm close()

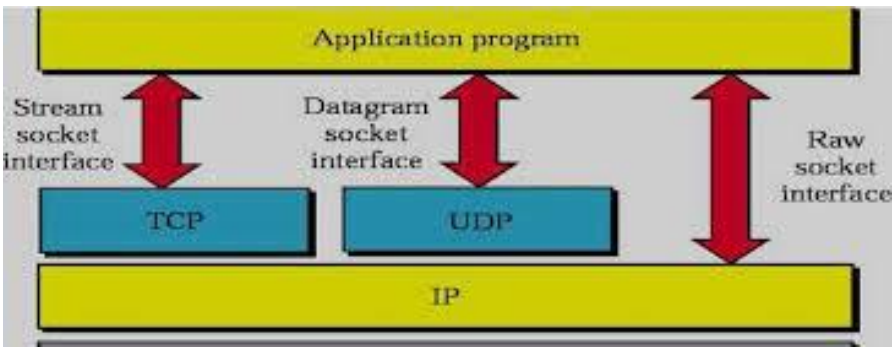
b. Datagram Socket

Hay còn gọi là socket phi kết nối, là socket hoạt động thông qua giao thức UDP (User Datagram Protocol). Datagram Socket có thể hoạt động kể cả khi không có sự thiết lập kết nối giữa 2 máy với nhau.



c. Raw

Raw Socket cung cấp cho người dùng quyền truy cập vào các giao thức truyền thông cơ bản, hỗ trợ trừu tượng hóa Socket. Các Socket này thường được định hướng datagram mặc dù các điểm điểm chính xác của chúng phụ thuộc vào giao diện được cung cấp bởi giao thức. Raw Socket không dành cho người dùng phổ thông mà chúng được cung cấp chủ yếu cho người quan tâm đến việc phát triển các giao thức truyền thông mới hoặc để có quyền truy cập vào một số phương tiện khó hiểu hơn của một giao thức hiện có.



4) So sánh phương thức Send() và SendTo()?

Send()	SendTo()

<table>
<tr><td>Gửi dữ liệu đến một socket đã kết nối</td><td>Gửi dữ liệu đến một EndPoint cụ thể</td></tr>
<tr><td>

Send(Byte[])
Send(IList<ArraySegment<Byte>>)
Send(ReadOnlySpan<Byte>)
Send(Byte[], SocketFlags)
Send(IList<ArraySegment<Byte>>, SocketFlags)
Send(ReadOnlySpan<Byte>, SocketFlags)
Send(Byte[], Int32, SocketFlags)
Send(IList<ArraySegment<Byte>>, SocketFlags, SocketError)
Send(ReadOnlySpan<Byte>, SocketFlags, SocketError)
Send(Byte[], Int32, Int32, SocketFlags)

</td><td>

SendTo(Byte[], EndPoint)
SendTo(Byte[], SocketFlags, EndPoint)
SendTo(Byte[], Int32, SocketFlags, EndPoint)
SendTo(Byte[], Int32, Int32, SocketFlags, EndPoint)

</td></tr>
<tr><td>

```
public static int
SendReceiveTest1(Socket server)
{
    byte[] msg =
Encoding.UTF8.GetBytes("This is a
test");
    byte[] bytes = new byte[256];
    try
    {
        // Blocks until send returns.
        int i = server.Send(msg);
        Console.WriteLine("Sent {0}
bytes.", i);

        // Get reply from the server.
        i = server.Receive(bytes);
```

</td><td>

```
public static void SendTo1()
{
    IPHostEntry hostEntry =
Dns.GetHostEntry(Dns.GetHostName());
    IPEndPoint endPoint = new
IPEndPoint(hostEntry.AddressList[0],
11000);

    Socket s = new
Socket(endPoint.Address.AddressFamily
,
        SocketType.Dgram,
        ProtocolType.Udp);

    byte[] msg =
Encoding.ASCII.GetBytes("This is a
test");
```

</td></tr>
</table>

```
Console.WriteLine(Encoding.UTF8.GetString(bytes));
    }
    catch (SocketException e)
    {
        Console.WriteLine("{0} Error code: {1}.", e.Message, e.ErrorCode);
        return (e.ErrorCode);
    }
    return 0;
}
```

```
    Console.WriteLine("Sending data.");
    // This call blocks.
    s.SendTo(msg, endPoint);
    s.Close();
}
```

5) So sánh phương thức Receive() và ReceiveFrom()?

Receive()	**ReceiveFrom()**
Receives data from a bound Socket.	Receives a datagram and stores the source endpoint.
➢ Receive(Byte[], Int32, Int32, SocketFlags, SocketError) ➢ Receive(Byte[], Int32, Int32, SocketFlags) ➢ Receive(Span<Byte>, SocketFlags, SocketError) ➢ Receive(IList<ArraySegment<Byte>>, SocketFlags, SocketError) ➢ Receive(Byte[], Int32, SocketFlags) ➢ Receive(IList<ArraySegment<Byte>>, SocketFlags) ➢ Receive(Byte[], SocketFlags) ➢ Receive(Span<Byte>) ➢ Receive(IList<ArraySegment<Byte>>)	➢ ReceiveFrom(Byte[], EndPoint) ➢ ReceiveFrom(Byte[], SocketFlags, EndPoint) ➢ ReceiveFrom(Byte[], Int32, SocketFlags, EndPoint) ➢ ReceiveFrom(Byte[], Int32, Int32, SocketFlags, EndPoint)

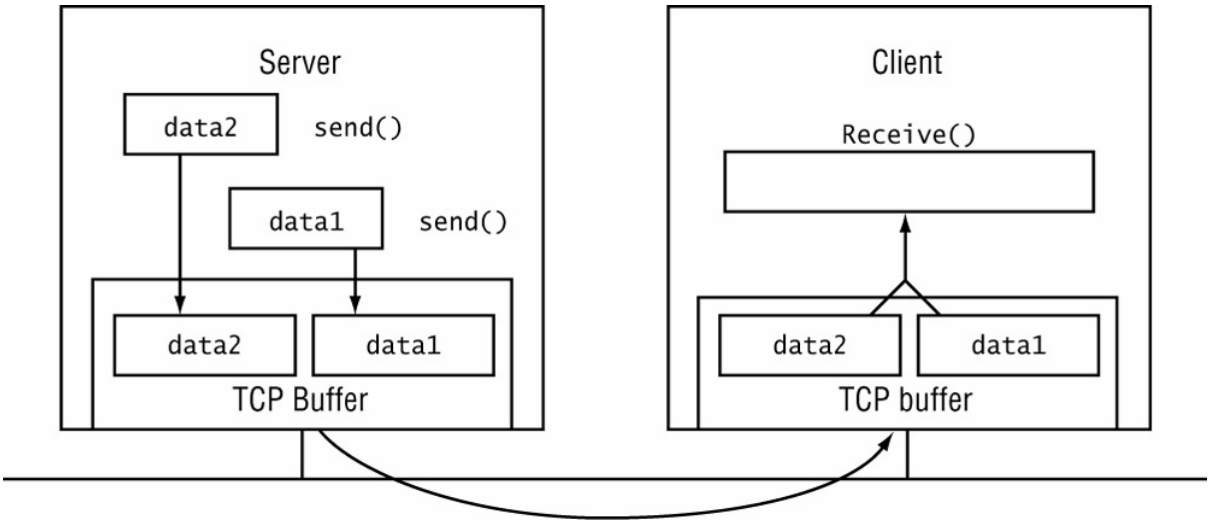
➢ Receive(Byte[]) ➢ Receive(Span<Byte>, SocketFlags)	

public static int SendReceiveTest2(Socket server)
{
byte[] msg = Encoding.UTF8.GetBytes("This is a test");
byte[] bytes = new byte[256];
try
{
// Blocks until send returns.
int byteCount = server.Send(msg, SocketFlags.None);
Console.WriteLine("Sent {0} bytes.", byteCount);
 	
// Get reply from the server.
byteCount = server.Receive(bytes, SocketFlags.None);
if (byteCount > 0)
 	
Console.WriteLine(Encoding.UTF8.GetString(bytes));
}
catch (SocketException e)
{
Console.WriteLine("{0} Error code: {1}.", e.Message, e.ErrorCode);
return (e.ErrorCode);
}
return 0;
}
	public static void ReceiveFrom2()
	{
	IPHostEntry hostEntry = Dns.GetHostEntry(Dns.GetHostName());
	IPEndPoint endPoint = new IPEndPoint(hostEntry.AddressList[0], 11000);
	
	Socket s = new Socket(endPoint.Address.AddressFamily,
	SocketType.Dgram,
	ProtocolType.Udp);
	
	// Creates an IpEndPoint to capture the identity of the sending host.
	IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
	EndPoint senderRemote = (EndPoint)sender;
	
	// Binding is required with ReceiveFrom calls.
	s.Bind(endPoint);
	
	byte[] msg = new Byte[256];
	Console.WriteLine ("Waiting to receive datagrams from client...");
	// This call blocks.
	s.ReceiveFrom(msg, SocketFlags.None, ref senderRemote);
	s.Close();
	}

6) Khi sử dụng socket hướng kết nối (TCP) thì phải lưu ý vấn đề gì?

Vấn đề về bộ đệm có kích thước nhỏ:

Hệ điều hành Window dùng bộ đệm TCP để gởi và nhận dữ liệu. Điều này là cần thiết để TCP có thể gởi lại dữ liệu bất cứ lúc nào cần thiết. Một khi dữ liệu đã được hồi báo nhận thành công thì nó mới được xóa khỏi bộ đệm.



Dữ liệu đến cũng được hoạt động theo cách tương tự. Nó sẽ ở lại trong bộ đệm cho đến khi phương thức Receive() được dùng để đọc nó. Nếu phương thức Receive() không đọc toàn bộ dữ liệu ở trong bộ đệm, phần còn lại vẫn được nằm ở đó và chờ phương thức Receive() tiếp theo được đọc. Dữ liệu sẽ không bị mất nhưng chúng ta sẽ không lấy được các đoạn dữ liệu mình mong muốn.

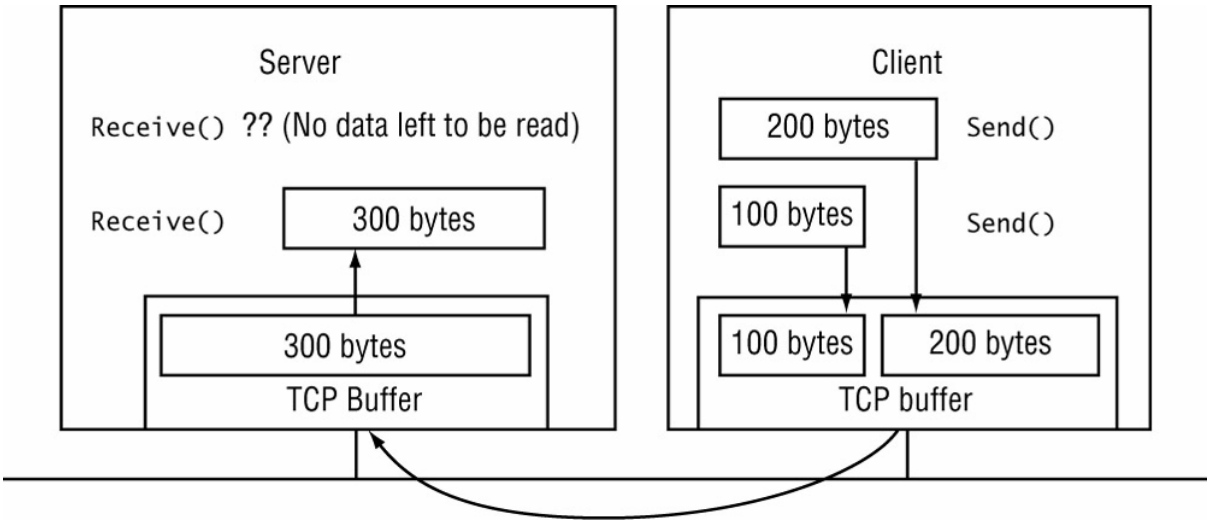
Bởi vì bộ đệm dữ liệu không đủ lớn để lấy hết dữ liệu ở bộ đệm TCP nên phương thức Receive() chỉ có thể lấy được một lượng dữ liệu có độ lớn đúng bằng độ lớn của bộ đệm dữ liệu, phần còn lại vẫn nằm ở bộ đệm TCP và nó được lấy khi gọi lại phương thức Receive(). Do đó câu chào Client của Server phải dùng tới hai lần gọi phương thức Receive() mới lấy được hết. Trong lần gởi và nhận dữ liệu kế tiếp, đoạn dữ liệu tiếp theo được đọc từ bộ đệm TCP do đó nếu ta gởi dữ liệu với kích thước lớn hơn 10 byte thì khi nhận ta chỉ nhận được 10 byte đầu tiên.

Bởi vì vậy nên trong khi lập trình mạng chúng ta phải quan tâm đến việc đọc dữ liệu từ bộ đệm TCP một cách chính xác. Bộ đệm quá nhỏ có thể dẫn đến tình trạng thông điệp nhận sẽ không khớp với thông điệp gởi, ngược lại bộ đệm quá lớn sẽ làm cho các thông

điệp bị trộn lại, khó xử lý. Việc khó nhất là làm sao phân biệt được các thông điệp được đọc từ Socket.

7) Trình bày các giải pháp tách biên thông diệp của socket hướng kết nối (TCP)?

Một trong những khó khăn của những nhà lập trình mạng khi sử dụng giao thức TCP để chuyển dữ liệu là giao thức này không quan tâm đến biên dữ liệu.



Như trên hình vấn đề xảy ra khi truyền dữ liệu là không đảm bảo được mỗi phương thức Send() sẽ không được đọc bởi một phương thức Receive(). Tất cả dữ liệu được đọc từ phương thức Receive() không thực sự được đọc trực tiếp từ mạng mà nó được đọc từ bộ đệm TCP. Khi các gói tin TCP được nhận từ mạng sẽ được đặt theo thứ tự trong bộ đệm TCP. Mỗi khi phương thức Receive() được gọi, nó sẽ đọc dữ liệu trong bộ đệm TCP, không quan tâm đến biên dữ liệu.

Giải quyết các vấn đề với thông điệp TCP:

Để giải quyết vấn đề với biên dữ liệu không được bảo vệ, chúng ta phải tìm hiểu một số kỹ thuật để phân biệt các thông điệp. Ba kỹ thuật thông thường dùng để phân biệt các thông điệp được gởi thông qua TCP:

- Luôn luôn sử dụng các thông điệp với kích thước cố định

Cách dễ nhất nhưng cũng là cách tốn chi phí nhất để giải quyết vấn đề với các thông điệp TCP là tạo ra các giao thức luôn luôn truyền các thông điệp với kích thước cố định. Bằng cách thiết lập tất cả các thông điệp có cùng kích thước, chương trình TCP nhận có thể biết toàn bộ thông điệp được gởi từ Client.

```
private static int SendData(Socket s, byte[] data)
```

```
{
    int total = 0;
    int size = data.Length;
    int dataleft = size;
    int sent;
    while (total < size)
    {
        sent = s.Send(data, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}

private static byte[] ReceiveData(Socket s, int size)
{
    int total = 0;
    int dataleft = size;
    byte[] data = new byte[size];
    int recv;
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
}
```

- Gởi kèm kích thước thông điệp cùng với mỗi thông điệp

Cách giải quyết vấn đề biên thông điệp của TCP bằng cách sử dụng các thông điệp với kích thước cố định là một giải pháp lãng phí bởi vì tất cả các thông điệp đều phải cùng kích thước. Nếu các thông điệp nào chưa đủ kích thước thì phải thêm phần đệm vào, gây lãng phí băng thông mạng.

Một giải pháp cho vấn đề cho phép các thông điệp được gởi với các kích thước khác nhau là gởi kích thước thông điệp kèm với thông điệp. Bằng cách này thiết bị nhận sẽ biết được kích thước của mỗi thông điệp.

```
private static int SendVarData(Socket s, byte[] buff)
{
    int total = 0;
    int size = buff.Length;
    int dataleft = size;
    int sent;
    byte[] datasize = new byte[4];
    datasize = BitConverter.GetBytes(size);
    sent = s.Send(datasize);
    while (total < size)
    {
        sent = s.Send(buff, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}

private static byte[] ReceiveVarData(Socket s)
{
    int total = 0;
    int recv;
    byte[] datasize = new byte[4];
    recv = s.Receive(datasize, 0, 4, 0);
    int size = BitConverter.ToInt32(datasize, 0);
    int dataleft = size;
    byte[] data = new byte[size];
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit ");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
```

```
    return data;
}
```

- Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

Một cách khác để gởi các thông điệp với kích thước khác nhau là sử dụng các hệ thống đánh dấu. Hệ thống này sẽ chia các thông điệp bởi các ký tự phân cách để báo hiệu kết thúc thông điệp. Khi dữ liệu được nhận từ Socket, dữ liệu được kiểm tra từng ký tự một để phát hiện các ký tự phân cách, khi các ký tự phân cách được phát hiện thì dữ liệu trước ký tự phân cách chính là một thông điệp và dữ liệu sau ký tự phân cách sẽ bắt đầu một thông điệp mới.

Phương pháp này có một số hạn chế, nếu thông điệp lớn nó sẽ làm giảm tốc độ của hệ thống vì toàn bộ các ký tự của thông điệp đều phải được kiểm tra. Cũng có trường hợp một số ký tự trong thông điệp trùng với các ký tự phân cách và thông điệp này sẽ bị tách ra thành các thông điệp con, điều này làm cho chương trình chạy bị sai lệch.

8) Khi sử dụng socket không kết nối (UDP) thì phải lưu ý vấn đề gì? Giải pháp?

Một trong những tính năng quan trọng của UDP mà TCP không có được đó là khả năng xử lý thông điệp mà không cần quan tâm đến biên thông điệp. UDP bảo vệ biên thông điệp của tất cả các thông điệp được gởi. Mỗi lần gọi phương thức ReceiveFrom() nó chỉ đọc dữ liệu được gởi từ một phương thức SendTo().

UDP Server:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)(sender);
```

```
        recv = newsock.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
        tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, tmpRemote);
        for (int i = 0; i < 5; i++)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref tmpRemote);
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        }
        newsock.Close();
    }
}
```

UDP Client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
        tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 1"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 2"), tmpRemote);
```

```
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 3"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 4"), tmpRemote);
            server.SendTo(Encoding.ASCII.GetBytes("Thong diep 5"), tmpRemote);
            Console.WriteLine("Dang dong client");
            server.Close();
        }
}
```

Vì UDP không quan tâm đến việc gởi lại các gói tin nên nó không dùng bộ đệm. Tất cả dữ liệu được gởi từ Socket đều được lập tức gởi ra ngoài mạng và tất cả dữ liệu được nhận từ mạng lập tức được chuyển cho phương thức ReceiveFrom() trong lần gọi tiếp theo. Khi phương thức ReceiveFrom() được dùng trong chương trình, các lập trình viên phải đảm bảo rằng bộ đệm phải đủ lớn để chấp nhận hết dữ liệu từ UDP Socket. Nếu bộ đệm quá nhỏ, dữ liệu sẽ bị mất.

Một khó khăn khác khi lập trình với giao thức UDP là khả năng bị mất gói tin bởi vì UDP là một giao thức phi kết nối nên không có cách nào mà thiết bị gởi biết được gói tin gởi có thực sự đến được đích hay không. Cách đơn giản nhất để ngăn chặn việc mất các gói tin là phải có cơ chế hồi báo giống như giao thức TCP. Các gói tin được gởi thành công đến thiết bị nhận thì thiết bị nhận phải sinh ra gói tin hồi báo cho thiết bị gởi biết đã nhận thành công. Nếu gói tin hồi báo không được nhận trong một khoảng thời gian nào đó thì thiết bị nhận sẽ cho là gói tin đó đã bị mất và gởi lại gói tin đó.

Có hai kỹ thuật dùng để truyền lại các gói tin UDP:

- Sử dụng Socket bất đồng bộ và một đối tượng Timer. Kỹ thuật này yêu cầu sử dụng một Socket bất đồng bộ mà nó có thể lắng nghe các gói tin đến không bị block. Sau khi Socket được thiết lập đọc bất đồng bộ, một đối tượng Timer có thể được thiết lập, nếu đối tượng Timer tắt trước khi hành động đọc bất đồng bộ kết thúc thì việc gởi lại dữ liệu diễn ra.
- Sử dụng Socket đồng bộ và thiết lập giá trị Socket time-out. Để làm được việc này, ta dùng phương thức SetSocketOption().

9) Trình bày cách sử dụng Socket Time-out?

Phương thức ReceiveFrom() là phương thức bị block. Nó sẽ block chương trình lại cho đến khi chương trình nhận dữ liệu. Nếu dữ liệu không bao giờ nhận, chương trình sẽ block mã mãi. Mặc định phương thức ReceiveFrom() sẽ bị block mãi mãi nếu không có dữ liệu được đọc. phương thức SetSocketOption() cung cấp nhiều tùy chọn cho các

Socket đã được tạo, một trong những tùy chọn đó là ReceiveTimeout. Nó sẽ thiết lập khoảng thời gian Socket sẽ chờ dữ liệu đến trước khi phát ra tín hiệu time-out.

Giá trị ban đầu của ReceiveTimeout được thiết lập là 0 cho biết nó sẽ chờ dữ liệu mãi mãi. Sau khi thêm phương thức SetSocketOpition() và được thiết lập giá trị 3000 mili giây thì hàm ReceiveFrom() sẽ đợi dữ liệu trong 3 giây, sau 3 giây nếu không có dữ liệu để đọc thì nó sẽ phát sinh ra biệt lệ do đó ta phải đặt hàm này trong khối try – catch để xử lý biệt lệ.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TimeoutUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        // Tạo Socket
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, P
rotocolType.Udp);
        // Lấy thông tin timeout ban đầu mặc định là 0 (chờ mãi mãi)
        int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket, Socket
OptionName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt);
        // Thiết đặt timeout là 3 giây, sau 3 giây, nếu không nhận được thông tin
gì gửi đến thì báo lỗi
        server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveT
imeout, 3000);
        sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket, SocketOpti
onName.ReceiveTimeout);
        Console.WriteLine("Gia tri timeout moi: {0}", sockopt);
        string welcome = "Xin chao server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
```

```
        data = new byte[1024];
        recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", mpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote);
            data = new byte[1024];
            recv = server.ReceiveFrom(data, ref tmpRemote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Exiting");
        server.Close();
    }
}
```

10) Trình bày ý nghĩa và cách sử dụng các lớp TCPListener, TCPClient và UDPClient của .Net?

a. TCPClient

Lớp TcpClient nằm ở namespace System.Net.Sockets được thiết kế để hỗ trợ cho việc viết các ứng dụng TCP Client được dễ dàng.

Lớp TcpClient cho phép tạo ra một đối tượng Tcp Client sử dụng một trong ba phương thức tạo lập sau:

- **TcpClient():** là phương thức tạo lập đầu tiên, đối tượng được tạo ra bởi phương thức tạo lập này sẽ gắn kết với một địa chỉ cục bộ và một port TCP ngẫu nhiên. Sau khi đối tượng TcpClient được tạo ra, nó phải được kết nối đến thiết bị ở xa thông qua phương thức Connect() như ví dụ dưới đây:

```
TcpClient newcon = new TcpClient();
newcon.Connect("192.168.6.1", 8000);
```

- **TcpClient(IPEndPoint localEP):** phương thức tạo lập này cho phép chúng ta chỉ ra địa chỉ IP cục bộ cùng với port được dùng. Đây là phương thức tạo lập thường

được sử dụng khi thiết bị có nhiều hơn một card mạng và chúng ta muốn dữ liệu được gởi trên card mạng nào. Phương thức Connect() cũng được dùng để kết nối với thiết bị ở xa:

```
IPEndPoint iep = new IPEndPoint(IPAddress,Parse("192.168.6.1"), 8000);
                TcpClient newcon = new TcpClient(iep);
                 newcon.Connect("192.168.6.2", 8000);
```

- **TcpClient(String host, int port):** phương thức tạo lập thứ ba này thường được sử dụng nhất, nó cho phép chỉ ra thiết bị nhận trong phương thức tạo lập và không cần phải dùng phương thức Connect(). Địa chỉ của thiết bị ở xa có thể là một chuỗi hostname hoặc một chuỗi địa chỉ IP. Phương thức tạo lập của TcpClient sẽ tự động phân giải hostname thành địa chỉ IP. Ví dụ:

```
TcpClient newcon = new TcpClient("www.isp.net", 8000);
```

Mỗi khi đối tượng TcpClient được tạo ra, nhiều thuộc tính và phương thức có thể được dùng để xử lý việc truyền dữ liệu qua lại giữa các thiết bị.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpClientSample
{
        public static void Main()
        {
                byte[] data = new byte[1024];
                string input, stringData;
                TcpClient server;
                try
                {
                        server = new TcpClient("127.0.0.1", 5000);
                }
                catch (SocketException)
                {
                        Console.WriteLine("Khong the ket noi den server");
                        return;
                }
                NetworkStream ns = server.GetStream();
                int recv = ns.Read(data, 0, data.Length);
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
```

```
            while (true)
            {
                input = Console.ReadLine();
                if (input == "exit")
                    break;
                ns.Write(Encoding.ASCII.GetBytes(input), 0,
                input.Length);
                ns.Flush();
                data = new byte[1024];
                recv = ns.Read(data, 0, data.Length);
                stringData = Encoding.ASCII.GetString(data, 0, recv);
                Console.WriteLine(stringData);
            }
            Console.WriteLine("Dang ngat ket noi voi server...");
            ns.Close();
            server.Close();
        }
}
```

b. TCPListener

Cũng giống như lớp TcpClient, lớp TcpListener cũng cho phép chúng ta tạo ra các chương trình TCP Server một cách đơn giản.

Lớp TcpListener có ba phương thức tạo lập:

- **TcpListener(int port)**: gắn một đối tượng TcpListener vào một port được chỉ ra trên máy cục bộ.
- **TcpListener(IPEndPoint ie)**: gắn một đối tượng TcpListener vào một đối tượng EndPoint cục bộ.
- **TcpListener(IPAddress addr, int port)**: gắn một đối tượng TcpListener vào một đối tượng IPAddress và một port.

Không giống như lớp TcpClient, các phương thức tạo lập của lớp TcpListener yêu cầu ít nhất một tham số: số port mà Server lắng nghe kết nối. Nếu Server có nhiều card mạng và ta muốn lắng nghe trên một card mạng nào đó thì ta có thể dùng một đối tượng IPEndPoint để chỉ ra địa chỉ IP của card cùng với số port dùng để lắng nghe.

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

```
{
        public static void Main()
        {
                int recv;
                byte[] data = new byte[1024];
                TcpListener newsock = new TcpListener(5000);
                newsock.Start();
                Console.WriteLine("Dan cho client ket noi den...");
                TcpClient client = newsock.AcceptTcpClient();
                NetworkStream ns = client.GetStream();
                string welcome = "Hello Client";
                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;
                        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
                        ns.Write(data, 0, recv);
                }
                ns.Close();
                client.Close();
                newsock.Stop();
        }
}
```

c. UDPClient

Lớp UdpClient được tạo ra để giúp cho việc lập trình mạng với giao thức UDP được đơn giản hơn. Lớp UdpClient có bốn phương thức tạo lập:

- **UdpClient()**: tạo ra một đối tượng UdpClient nhưng không gắn vào bất kỳ địa chỉ hay port nào.
- **UdpClient(int port)**: gắn đối tượng UdpClient mới tạo vào một port.
- **UdpClient(IPEndPoint iep)**: gắn đối tượng UdpClient mới tạo vào một địa chỉ IP cục bộ và một port.
- **UdpClient(string host, int port)**: gắn đối tượng UdpClient mới tạo vào một địa chỉ IP và một port bất kỳ và kết hợp nó với một địa chỉ IP và port ở xa.

UDPClient phía server:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpSrvrSample
{
	public static void Main()
	{
		byte[] data = new byte[1024];
		IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
		UdpClient newsock = new UdpClient(ipep);
		Console.WriteLine("Dang cho client ket noi den...");
		IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
		data = newsock.Receive(ref sender);
		Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
		Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
		string welcome = "Xin chao client";
		data = Encoding.ASCII.GetBytes(welcome);
		newsock.Send(data, data.Length, sender);
		while (true)
		{
			data = newsock.Receive(ref sender);
			Console.WriteLine(Encoding.ASCII.GetString(data, 0,
data.Length));
			newsock.Send(data, data.Length, sender);
		}
	}
}
```

UDPClient phía client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpClientSample
{
	public static void Main()
	{
		byte[] data = new byte[1024];
		string input, stringData;
		UdpClient server = new UdpClient("127.0.0.1", 5000);
		IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
```

```
            string welcome = "Xin chao server";
            data = Encoding.ASCII.GetBytes(welcome);
            server.Send(data, data.Length);
            data = server.Receive(ref sender);
            Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
            stringData = Encoding.ASCII.GetString(data, 0, data.Length);
            Console.WriteLine(stringData);
            while (true)
            {
                input = Console.ReadLine();
                if (input == "exit")
                    break;
                server.Send(Encoding.ASCII.GetBytes(input), input.Length);
                data = server.Receive(ref sender);
                stringData = Encoding.ASCII.GetString(data, 0, data.Length);
                Console.WriteLine(stringData);
            }
            Console.WriteLine("Dang dong client");
            server.Close();
        }
}
```

11) Trình bày cách sử dụng NetworkStream

Điều khiển thông điệp dùng giao thức TCP thường gây ra khó khăn cho các lập trình viên nên .NET Framework cung cấp một số lớp để giảm gánh nặng lập trình. Một trong những lớp đó là NetworkStream, và hai lớp dùng để gởi và nhận text sử dụng giao thức TCP là StreamWriter và StreamReader.

Lớp NetworkStream nằm trong namespace System.Net.Socket, lớp này có nhiều phương thức tạo lập để tạo một thể hiện của lớp NetworkStream nhưng phương thức tạo lập sau hay được dùng nhất:

```
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                           ProtocolType.Tcp);
NetworkStream ns = new NetworkStream(server);
```

TCP Client sử dụng Network Stream:

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

```
class NetworkStreamTcpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 500);
        Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stre
am, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        if (ns.CanRead)
        {
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        else
        {
            Console.WriteLine("Error: Can't read from this Socket");
            ns.Close();
            server.Close();
            return;
        }
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
```

```
                break;
            if (ns.CanWrite)
            {
                ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
                ns.Flush();
            }
            recv = ns.Read(data, 0, data.Length);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang ngat ket noi voi server...");
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

12) Lớp StreamReader và StreamWriter dùng để làm gì?

Namespace System.IO chứa hai lớp StreamReader và StreamWriter điều khiển việc đọc và ghi các thông điệp text từ mạng. Cả hai lớp đều có thể được triển khai với một đối tượng NetworkStream để xác định các hệ thống đánh dấu cho các thông điệp TCP.

```
public StreamReader(Stream stream);
public StreamWriter(Stream stream);
```

TCP Client:

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpClient
{
    public static void Main()
    {
        string data;
        string input;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        try
```

```
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        data = sr.ReadLine();
        Console.WriteLine(data);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            sw.WriteLine(input);
            sw.Flush();
            data = sr.ReadLine();
            Console.WriteLine(data);
        }
        Console.WriteLine("Dang dong ket noi voi server...");
        sr.Close();
        sw.Close();
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

TCP Server:

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

class StreamTcpSrvr
{
```

```
    public static void Main()
    {
        string data;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi toi...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
        newclient.Address, newclient.Port);
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        string welcome = "Hello Client";
        sw.WriteLine(welcome);
        sw.Flush();
        while (true)
        {
            try
            {
                data = sr.ReadLine();
            }
            catch (IOException)
            {
                break;
            }
            Console.WriteLine(data);
            sw.WriteLine(data);
            sw.Flush();
        }
        Console.WriteLine("Da dong ket noi voi Client {0}", newclient.Address);
        sw.Close();
        sr.Close();
        ns.Close();
    }
}
```

13) Phương thức BlockCopy() dùng để làm gì? Cho ví dụ?

Sao chép một số byte được chỉ định từ một mảng nguồn (src) bắt đầu từ một offset cụ thể thành một mảng đích (dst) bắt đầu từ một offset cụ thể.

```
public static void BlockCopy (Array src, int srcOffset, Array dst, int dstOffset,
int count);
```

Ví dụ:

```
using System;

class Program
{
    static void Main()
    {
        int[] array1 = new int[5] { 9, 8, 7, 6, 5 };
        int[] array2 = new int[5] { 1, 2, 3, 4, 5 };
        int[] array3 = new int[array1.Length + array2.Length];
        Buffer.BlockCopy(array1, 0, array3, 0, array1.Length * sizeof(int));
        Buffer.BlockCopy(array2, 0, array3, array1.Length * sizeof(int), array
2.Length * sizeof(int));
        Console.WriteLine("PHAN TU CUA MANG THU 3 SAU KHI HOP NHAT 2 MANG :");
        foreach (int value in array3)
        {
            Console.WriteLine(value);
        }
        Console.ReadLine();
    }
}
```

Kết quả:

Array3 = 9, 8, 7, 6, 5, 5, 4, 3, 2, 1

14) Liệt kê một số cách thức tạo tiểu trình trong .Net

Thư viện lớp .NET Framework cung cấp một số phương pháp tạo tiểu trình mới:

- Thực thi một phương phức bằng tiểu trình trong Thread-pool
- Thực thi phương thức một cách bất đồng bộ
- Thực thi một phương thức bằng tiểu trình theo chu kỳ hay ở một thời điểm xác định
- Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle

15) So sánh cách thức tạo tiểu trình mới và ThreadPool?

CLASS NAME	DESCRIPTION
Mutex	It is a synchronization primitive that can also be used for IPS (interprocess synchronization).
Monitor	This class provides a mechanism that access objects in synchronize manner.
Semaphore	This class is used to limit the number of threads that can access a resource or pool of resources concurrently.
Thread	This class is used to creates and controls a thread, sets its priority, and gets its status.
ThreadPool	This class provides a pool of threads that can be used to execute tasks, post work items, process asynchronous I/O, wait on behalf of other threads, and process timers.
ThreadLocal	This class provides thread-local storage of data.
Timer	This class provides a mechanism for executing a method on a thread pool thread at specified intervals. You are not allowed to inherit this class.
Volatile	This class contains methods for performing volatile memory operations.

16) Ủy nhiệm hàm (Delegate) là gì? Cho ví dụ?

Một delegate là một lớp nó giữ tham chiếu đến một phương thức mà phương thức này điều khiển sự kiện được nhận. Delegate trong C# là tương tự như con trỏ tới các hàm, trong C hoặc trong C++. Delegate là một biến kiểu tham chiếu chứa tham chiếu tới một phương thức. Tham chiếu đó có thể được thay đổi tại runtime.

```
public delegate void MyDelegate(string msg); // khai báo ủy nhiệm
```

```
// gán ủy nhiệm cho phương thức MethodA
MyDelegate d = new MyDelegate(MethodA);
// MyDelegate d = MethodA;
// MyDelegate d = (string msg) =>  Console.WriteLine(msg);

// Định nghĩa MethodA
static void MethodA(string message)
{
    Console.WriteLine(message);
}
```

Khi sử dụng, ta chỉ cần gọi `d.Invoke("Hello World!");` giống với việc ta gọi trực tiếp MethodA vậy.

17) Trình bày cách thức tạo tiểu trình bất đồng bộ?

- Phương thức **Blocking**: dừng quá trình thực thi của tiểu trình hiện hành cho đến khi phương thức thực thi bất đồng bộ kết thúc. Điều này rất giống với sự thực thi đồng bộ. Tuy nhiên, nếu linh hoạt chọn thời điểm chính xác để đưa mã lệnh vào trạng thái dừng (block) thì vẫn còn cơ hội thực hiện thêm một số việc trước khi mã lệnh đi vào trạng thái này.
- Phương thức **Polling**: lặp đi lặp lại việc kiểm tra trạng thái của phương thức thực thi bất đồng bộ để xác định nó kết thúc hay chưa. Đây là một kỹ thuật rất đơn giản, nhưng nếu xét về mặt xử lý thì không được hiệu quả. Nên tránh các vòng lặp chặt làm lãng phí thời gian của bộ xử lý; tốt nhất là nên đặt tiểu trình thực hiện polling vào trạng thái nghỉ (sleep) trong một khoảng thời gian bằng cách sử dụng Thread.Sleep giữa các lần kiểm tra trạng thái. Bởi kỹ thuật polling đòi hỏi phải duy trì một vòng lặp nên hoạt động của tiểu trình chờ sẽ bị giới hạn, tuy nhiên có thể dễ dàng cập nhật tiến độ công việc.
- Phương thức **Waiting**: sử dụng một đối tượng dẫn xuất từ lớp System.Threading.WaitHandle để báo hiệu khi phương thức thực thi bất đồng bộ kết thúc. Waiting là một cải tiến của kỹ thuật polling, nó cho phépchờ nhiều phương thức thực thi bất đồng bộ kết thúc. Có thể chỉ định các giá trị time-out cho phép tiểu trình thực hiện waiting dừng lại nếu phương thức thực thi bất đồng bộ đã diễn ra quá lâu, hoặc muốn cập nhật định kỳ bộ chỉ trạng thái.

- Phương thức **Callback**: Callback là một phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Mã lệnh thực hiện lời gọi không cần thực hiện bất kỳ thao tác kiểm tra nào, nhưng vẫn có thể tiếp tục thực hiện các công việc khác. Callback rất linh hoạt nhưng cũng rất phức tạp, đặc biệt khi có nhiều phương thức thực thi bất đồng bộ chạy đồng thời nhưng sử dụng cùng một callback. Trong những trường hợp như thế, phải sử dụng các đối tượng trạng thái thích hợp để so trùng các phương thức đã hoàn tất với các phương thức đã khởi tạo.

18) Liệt kê và trình bày các cách thức để xác định một phương thức thực thi bất đồng bộ đã kết thúc.

Để kiểm tra một tiểu trình đã kết thúc hay chưa là kiểm tra thuộc tính Thread.IsAlive. Thuộc tính này trả về true nếu tiểu trình đã được khởi chạy nhưng chưa kết thúc hay bị hủy.

Thông thường,cần một tiểu trình để đợi một tiểu trình khác hoàn tất việc xử lý của nó. Thay vì kiểm tra thuộc tính IsAlive trong một vòng lặp, có thể sử dụng phương thức Thread.Join. Phương thức này khiến tiểu trình đang gọi dừng lại (block) cho đến khi tiểu trình được tham chiếu kết thúc.

Có thể tùy chọn chỉ định một khoảng thời gian (giá trị int hay TimeSpan) mà sau khoảng thời gian này, Join sẽ hết hiệu lực và quá trình thực thi của tiểu trình đang gọi sẽ phục hồi lại. Nếu chỉ định một giá trị time-out, Join trả về true nếu tiểu trình đã kết thúc, và false nếu Join đã hết hiệu lực.

19) Socket bất đồng bộ là gì? Vì sao cần sử dụng Socket bất đồng bộ?

Trong các chương trước, chúng ta đã lập trình Socket trong chế độ blocking. Socket ở chế độ blocking sẽ chờ mãi mãi cho đến khi hoàn thành nhiệm vụ của nó. Trong khi nó bị blocking thì các chức năng khác của chương trình không thực hiện được.

Khi lập trình Windows thì lúc gọi một phương thức bị blocking thì toàn bộ chương trình sẽ đứng lại và không thực hiện các chức năng khác được. Do đó việc lập trình bất đồng bộ là cần thiết để cho chương trình khỏi bị đứng.

20) Trình bày ý nghĩa và ví dụ về cách sử dụng các phương thức Socket bất đồng bộ:

Phương thức được dùng để thành lập kết nối với thiết bị ở xa phụ thuộc vào chương trình đóng vai trò là Server hay Client. Nếu là Server thì phương thức BeginAccept() được dùng, nếu là Client thì phương thức BeginConnect() được dùng.

a. BeginAccept() và EndAccept()

BeginAccept() bắt đầu một phương thức bất đồng bộ để chấp nhận một kết nối đến.

```
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);
```

Sau khi phương thức BeginAccept() kết thúc, phương thức AsyncCallback định nghĩa sẽ được gọi khi kết nối xảy ra. Phương thức AsyncCallback phải bao gồm phương thức EndAccept() để kết thúc việc chấp nhận Socket.

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

b. BeginConnect() và EndConnect()

Để ứng dụng Client kết nối đến Server ở xa bằng phương thức bất đồng bộ ta phải dùng phương thức BeginConnect().

```
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);
```

Khi kết nối được thành lập phương thức do delegate AsyncCallback tham chiếu tới được gọi. Phương thức này dùng phương thức EndConnect() để hoàn thành việc kết nối.

```
public static void Connected(IAsyncResult iar)
{
Socket sock = (Socket)iar.AsyncState;
try
{
```

```
sock.EndConnect(iar);
} catch (SocketException)
{
Console.WriteLine("Unable to connect to host");
}
}
```

c. BeginSend() và EndSend()

Phương thức BeginSend() cho phép gởi dữ liệu đến một Socket đã được kết nối.

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

Phương thức EndSend() trả về số byte được gởi thành công thừ Socket. Sau đây là một ví dụ của phương thức EndSend():

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

d. BeginSendTo() và EndSendTo()

Phương thức BeginSendTo() được dùng với Socket phi kết nối để bắt đầu truyền tải dữ liệu bất đồng bộ tói thiết bị ở xa.

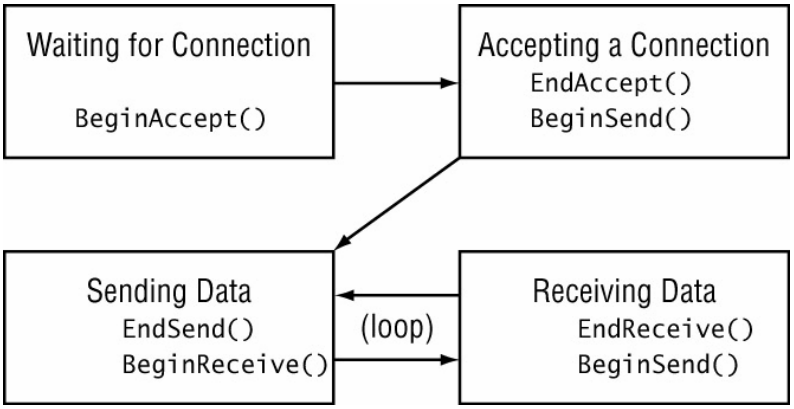
```
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.6"),
9050);
sock.BeginSendTo(data, 0, data.Length, SocketFlags.None, iep,
new AsynCallback(SendDataTo), sock);
```

Phương thức EndSendTo() sẽ được thực thi khi quá trình gởi dữ liệu kết thúc.

Đối số truyền vào của phương thức EndSendTo() là một đối tượng IAsyncResult, đối tượng này mang giá trị được truyền từ phương thức BeginSendTo(). Khi quá trình gởi dữ liệu bất đồng bộ kết thúc thì phương thức EndSendTo() sẽ trả về số byte mà nó thực sự gởi được.

e. BeginReceive() và EndReceive()

Cách sử dụng của các phương thức này cũng tương tự như cách sử dụng của các phương thức: BeginSend(), EndSend(), BeginSendTo() và EndSendTo().



f. BeginReceiveTo() và EndReceiveTo()