**LAB 3**

# I. Mục đích

Lập trình client – server sử dụng giao thức UDP

Nắm được các lỗi thường xảy ra khi lập trình theo giao thức UDP và cách hạn chế các lỗi này

# II. Yêu cầu

1) Xây dựng chương trình UDP Client – Server đơn giản (xem hướng dẫn)

# III. Hướng dẫn

### III.1. Xây dựng chương trình UDP Client – Server đơn giản

##### III.1.1. Hướng dẫn lập trình UDP Server

Để lập trình socket ta sử dụng 2 namespace:

using System.Net;

using System.Net.Sockets;

Tạo Server EndPoint, EndPoint này sẽ tham chiếu đến địa chỉ IP và Port của Server:

IPEndPoint serverEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);

Tạo Server Socket, Socket này dùng để trao đổi dữ liệu với client

Socket serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram,

ProtocolType.Udp);

Chương trình UDP Server khác với chương trình TCP Server ở chỗ nó không lắng nghe kết nối, trên socket ta chỉ việc Bind nó với Server EndPoint

serverSocket.Bind(serverEndPoint);

Khi client kết nối tới nó sẽ hiển thị thông tin của client đang kết nối đến:

Console.WriteLine(remote.ToString());

Để nhận dữ liệu từ client gởi lên ta dùng hàm ReceiveFrom với chú ý EndPoint chứa thông tin của client kết nối đến phải được truyền tham chiếu

serverSocket.ReceiveFrom(buff, 0, buff.Length,SocketFlags.None, ref remote);

##### III.1.2. Hướng dẫn lập trình UDP Client

Tạo Server Socket, Socket này sẽ được dùng để gởi dữ liệu tới Server

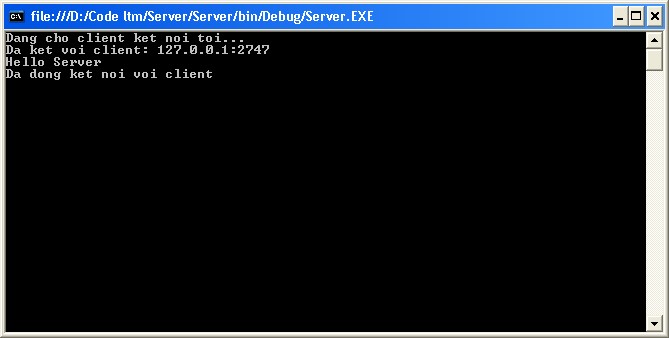
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

Gởi câu chào lên server, câu chào này sẽ được đặt trong mảng buff

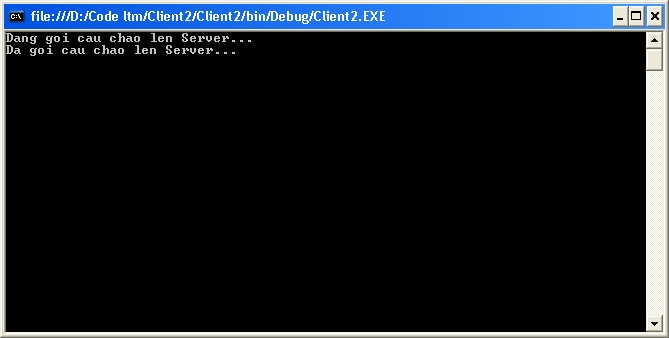
server.SendTo(buff, buff.Length, SocketFlags.None, serverEndPoint);

##### III.1.3. Test chương trình:

Chạy chương trình Server:



Chạy chương trình ở client:



##### III.1.4. Trả lời câu hỏi:

1. Như hình trên port 2747 ở đâu ra

Port 2747 là một cổng tạm thời (ephemeral port) do hệ điều hành phía client tự động cấp phát khi client tạo socket để gửi dữ liệu đến server. Khi một ứng dụng client cần kết nối mạng nhưng không chỉ định một cổng cụ thể, hệ điều hành sẽ chọn một cổng ngẫu nhiên từ một dải cổng được định sẵn để sử dụng cho phiên giao tiếp đó.

1. Có phải lúc nào client cũng mở port 2747 để kết nối với Server không ?

Không. Con số 2747 chỉ là một ví dụ tại thời điểm chương trình được chạy để chụp ảnh màn hình. Mỗi lần client khởi chạy lại, hệ điều hành có thể sẽ cấp phát một số hiệu cổng tạm thời khác (ví dụ: 2748, 51000, v.v.). Do đó, cổng này không cố định.

1. Tại sao khi lập trình mạng dùng giao thức UDP thì client phải gởi câu chào lên server trước?

Giao thức UDP là một giao thức không kết nối (connectionless). Điều này có nghĩa là không có một kết nối bền vững nào được thiết lập giữa client và server.

- UDP Server chỉ "lắng nghe" trên một cổng và không hề biết đến sự tồn tại của bất kỳ client nào cho đến khi nó nhận được một gói tin.

- Client phải gửi gói tin đầu tiên ("câu chào" hoặc bất kỳ dữ liệu nào) để server có thể biết được địa chỉ IP và số hiệu cổng của client đó.

- Thông tin này được trích xuất từ chính gói tin mà client gửi đến (thông qua phương thức ReceiveFrom). Sau khi nhận được thông tin này, server mới biết địa chỉ để gửi phản hồi trở lại cho client. Nếu không có gói tin đầu tiên từ client, server sẽ không có cách nào để liên lạc.

### III.2. Bài tập

1) Cải tiến chương trình để ở client gõ “exit” thì đóng client, khi client gõ “exit all” thì đóng cả client và server.

### III.3. Cải tiến chương trình UDP client-server để có thể gởi và nhận dữ liệu liên tục

##### III.3.1. Hướng dẫn lập trình UDP Client

Dùng vòng lặp vô hạn để khi người dùng nhập dữ liệu xong thì sẽ gởi lên server và chờ server gởi kết quả trả vể, lấy kết quả đó hiển thị lên màn hình

while (true)

{

str = Console.ReadLine(); buff = Encoding.ASCII.GetBytes(str);

serverSocket.SendTo(buff, 0, buff.Length, SocketFlags.None, remote); byteReceive = serverSocket.ReceiveFrom(buff, 0, buff.Length, SocketFlags.None, ref remote);

str = Encoding.ASCII.GetString(buff, 0, byteReceive);

Console.WriteLine(str);

}

##### III.3.2. Hướng dẫn lập trình UDP Server

Dùng vòng lặp vô hạn lấy kết quả client gởi lên, chuyển nó thành chuỗi và hiển thị lên màn hình đồng thời gởi lại dữ liệu nhận được về lại cho client

while (true)

{

buff = new byte[1024];

byteReceive = serverSocket.ReceiveFrom(buff, 0, buff.Length, SocketFlags.None, ref remote);

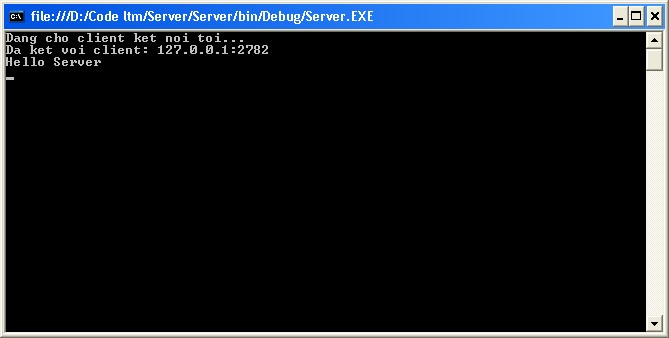
str = Encoding.ASCII.GetString(buff, 0, byteReceive); Console.WriteLine(str);

serverSocket.SendTo(buff, 0, buff.Lengh, SocketFlags.None, remote);

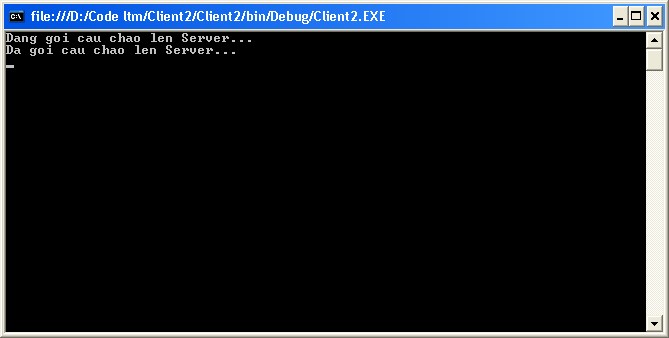
}

##### III.3.3. Test chương trình

Chạy chương trình ở server:



Chạy chương trình ở client:



Nhập thông điệp từ client và gởi lên server, lỗi sẽ xảy ra, hãy sửa lại cho hết lỗi

##### III.3.4. Trả lời câu hỏi:

1. Khi chạy chương trình với đoạn code gởi nhận dữ liệu như trên, lúc chưa nhập dữ liệu cho client để gởi lên thì không xảy ra lỗi nhưng khi nhập dữ liệu để gởi lên server sẽ xảy ra lỗi, vì sao lại xảy ra lỗi này ?

Đây là lỗi bên phía Client do biến remote được sử dụng trước khi được khởi tạo giá trị.

- Trước khi nhập tin nhắn: chương trình dừng để chờ nhập dữ liệu 🡪 lệnhserverSocket.SendTo(...) chưa được thực thi và không có lỗi nào xảy ra.

- Sau khi nhập tin nhắn: chương trình gọi lệnh serverSocket.SendTo(...), nhưng biến remote chưa có giá trị (null hoặc chưa được gán) 🡪 việc gửi dữ liệu đến một địa chỉ không xác định sẽ gây lỗi

1. Khi server chưa bật thì chương trình trên có bị lỗi không? Tại sao?

Có, client không báo lỗi ngày lập tực nhưng sẽ bị treo hoặc đơ vô thời hạn, vì:

- UDP là một giao thức không kết nối (connectionsless): giao thức chỉ đơn giản là gửi một gói tin vào mạng tới địa chỉ đích mà không cần biết có ai nhận hay không 🡪 thao tác gửi đi sẽ thành công mà không gây lỗi ngay cả khi server không mở.

- Bị khóa ở lệnh ReceiveFrom(): đây là lệnh chặn (blocking call), client sẽ dừng lại và chờ phản hồi từ server 🡪 vì server không được bật, sẽ không có phản hồi nào từ server được gửi cho client 🡪 client bị treo ở trạng thái chờ không thoát ra được.

1. Khi đang chạy chương trình tắt client thì chương trình trên có bị lỗi không? Tại sao

Không, server vẫn sẽ hoạt động bình thường, vì:

- Server UDP hoạt động theo nguyên tắc không kết nối, vòng lặp while (true) trong chương trình chỉ có nhiệm vụ nhận gói tin từ cổng bất kỳ.

- Không có kết nối nào được thiết lập bền vững giữa client và server 🡪 việc client ngắt kết nối không ảnh hưởng tới server, server chỉ tiếp tục trang thái chờ kết nối.

### III.4. Sử dụng phương thức Connect ở client để thiết lập kết nối trước với server

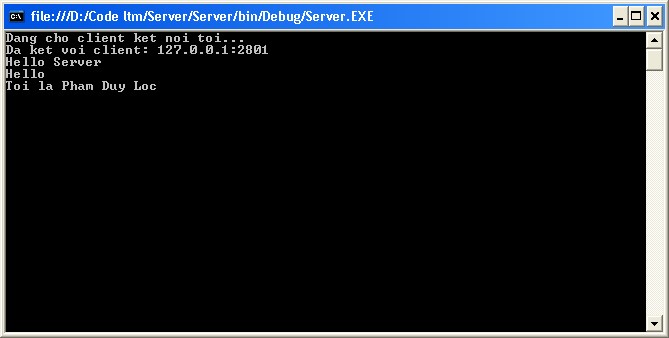
##### III.4.1. Hướng dẫn lập trình

Việc này được thực hiện bằng cách sử dụng phương thức Connect() ở lớp socket serverSocket.Connect(remote);

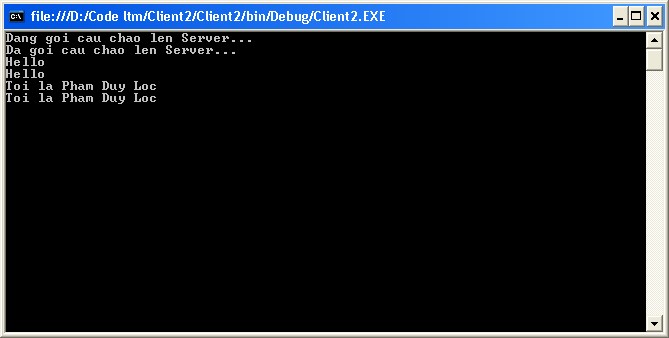
Sau khi gọi phương thức Connect() xong, ta có thể lập trình giống như lập trình với giao thức UDP với client

##### III.4.2. Test chương trình

Chạy chương trình ở server:



Chạy chương trình ở client:



##### III.4.3. Trả lời câu hỏi:

1. Khi chạy chương trình mà server chưa được bật thì có hiện tượng gì xảy ra? Tại sao lại có hiện tượng này?

Hiện tượng: khi client kết nối đến một server chưa được bật, chương trình không báo lỗi lập tức. Tuy nhiên khi client cố gắng nhận dữ liệu từ server, chương trình sẽ ném ra một ngoại lệ SocketException và thông báo lỗi.

Hiện tượng này xuất hiện vì:

- Phương thức Connect() trên socket UDP không thiết lập kết nối hai chiều bền vững như TCP. Nó chỉ thực hiện liên kết socket của client đến một địa chỉ server mặc định.

- Lệnh Connect() không gửi gói tin nào để “bắt tay” (handshake) với server, do đó không thể biết được server có hoạt động hay không.

- Khi client gửi gói tin tới server nhưng lại không có tiến trình nào lắng nghe ở cổng trên server, hệ điều hành của máy chủ sẽ gửi lại một gói tin ICMP (Internet Control Message Protocol) với thông báo “Destination Unreachable” (không thể đến đích) đến client.

1. Khi đang chạy tắt server thì chương trình trên có bị lỗi không? Tại sao

Có, chương trình client sẽ bị lỗi giống trường hợp ở câu hỏi 1, nhưng chỉ xảy ra khi client thực hiện gửi hoặc nhận sau khi server đã bị ngắt.

- Khi client và server đang giao tiếp, thực hiện các hành động gửi và nhận mà server bị ngắt đột ngột thì client sẽ không lập tức biết server bị ngắt.

- Nhưng khi client gửi gói tin tiếp theo tới server thì sẽ gặp tình trạng giống câu hỏi 1 (không có tiến trình lắng nghe bên server), hệ điều hành của server (nếu còn hoạt động) sẽ gửi một gói tin ICMP đến client.

- Client nhận được gói tin ICMP và nếu còn thực hiện hành động thì sẽ được thông báo rằng kết nối này đã mất.

1. Khi đang chạy chương trình tắt client thì chương trình trên có bị lỗi không? Tại sao

Không, server vẫn tiếp tục chạy bình thường.

- Server UDP không duy trì trang thái kết nối với client cụ thể nào. Server chỉ ở trong một vòng lặp chờ đợi gói tin đến với cổng của server.

- Với server, việc client ngắt kết nối mà không có thông báo thì chỉ là server không nhận thêm gói tin nào từ client đó nữa. Server vẫn tiếp tục chờ để phục vụ các gói tin từ các client khác.

### III.5. Kiểm tra khả năng phân biệt biên thông điệp của giao thức UDP

##### III.5.1. Hướng dẫn lập trình UDP Client

Thay đoạn code gởi và nhận thông điệp vô hạn bằng đoạn code gởi 5 thông điệp phân biệt lên server

for (int i = 1; i <= 5; i++)

{

buff = Encoding.ASCII.GetBytes("Thong Diep " + i.ToString()); serverSocket.SendTo(buff, 0, buff.Length, SocketFlags.None, remote); }

##### III.5.2. Hướng dẫn lập trình UDP Server

Thay đoạn code gởi và nhận thông điệp vô hạn bằng đoạn code nhận 5 thông điệp trên server

for (int i = 1; i <= 5; i++)

{

byteReceive = serverSocket.ReceiveFrom(buff, 0, buff.Length, SocketFlags.None, ref remote);

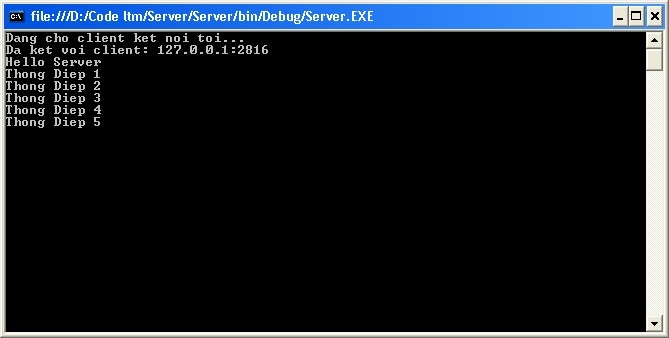
str = Encoding.ASCII.GetString(buff, 0, byteReceive);

Console.WriteLine(str);

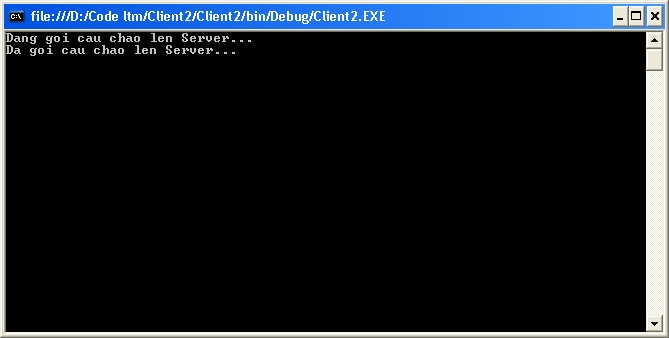
}

##### III.5.3. Test chương trình

Chạy chương trình server:



Chạy chương trình client:



#### III.5.4. Trả lời câu hỏi:

1) Tại sao khi lập trình bằng giao thức UDP các thông điệp được phân biệt với nhau ?

Vì mỗi thông điệp đước gửi đi dưới dang một gói tin (datagram) độc lập và có ranh giới với các gói tin khác, khác biệt hoàn toàn với TCP

- UDP là giao thức hướng thông điệp (Message-Oriented Protocol): Khi một ứng dụng gửi một khối dữ liệu (một thông điệp) qua socket UDP, giao thức sẽ đóng gói toàn bộ khối dữ liệu đó vào một datagram duy nhất và gửi đi. UDP không chia nhỏ hay gộp các thông điệp lại với nhau. Nếu bạn thực hiện lệnh SendTo() 5 lần, thì sẽ có chính xác 5 datagram được gửi đi trên mạng.

- Bảo toàn ranh giới thông điệp (Preservation of Message Boundaries): Phía nhận, khi thực hiện lệnh ReceiveFrom(), sẽ nhận được toàn bộ nội dung của một datagram trong một lần đọc. Nếu Client gửi một gói tin 50 byte và sau đó là một gói tin 100 byte, Server sẽ nhận được đúng một gói 50 byte trong lần đọc đầu tiên và một gói 100 byte trong lần đọc thứ hai. Nó không bao giờ nhận được 75 byte hay gộp cả hai thành 150 byte. Chính nhờ đặc điểm này mà các thông điệp riêng lẻ được phân biệt một cách tự nhiên.

## III.6. Ngăn cản mất dữ liệu khi lập trình mạng sử dụng giao thức UDP

Trong đoạn code gởi và nhận thông điệp của chương trình UDP Client đơn giản ở trên ta thay đổi kích thước bộ đệm như sau:

while (true)

{

str = Console.ReadLine(); buff = Encoding.ASCII.GetBytes(str);

serverSocket.SendTo(buff, 0, buff.Length, SocketFlags.None, remote); **buff = new byte[10];**

byteReceive = serverSocket.ReceiveFrom(buff, 0, buff.Length, SocketFlags.None, ref remote);

str = Encoding.ASCII.GetString(buff, 0, byteReceive);

Console.WriteLine(str);

}

#### III.6.1. Trả lời câu hỏi:

1) Khi gởi dữ liệu với kích thước lớn hơn 10 byte thì có lỗi xảy ra không? Tại sao?

Không có lỗi nhưng dữ liệu sẽ bị mất.

- Khi server gửi đi một thông điệp lớn hơn 10 byte thì khi client nhận gói tin UDP từ mạng nhưng vì bộ nhớ đệm chỉ có 10 byte nên client chỉ lấy 10 byte đầu tiên.

- Thao tác này không gây ra lỗi 🡪 client chỉ nhận và hiển thị được một phần của thông điệp, phần còn lại sẽ bị mất và không có bất kỳ cảnh báo nào.

#### III.6.2. Hướng dẫn lập trình để ngăn cản mất dữ liệu

Thay đoạn code gởi và nhận thông điệp vô hạn bằng đoạn code gởi 5 thông điệp phân biệt lên server

while (true)

{

input = Console.ReadLine(); if (input == "exit") break;

server.SendTo(Encoding.ASCII.GetBytes(input), tmpRemote); data = new byte[i]; try {

recv = server.ReceiveFrom(data, ref tmpRemote); stringData = Encoding.ASCII.GetString(data, 0, recv);

Console.WriteLine(stringData);

}

catch (SocketException)

{

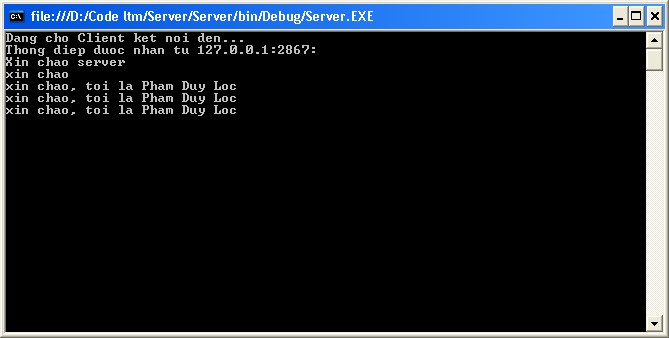
Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai"); i += 10;

}

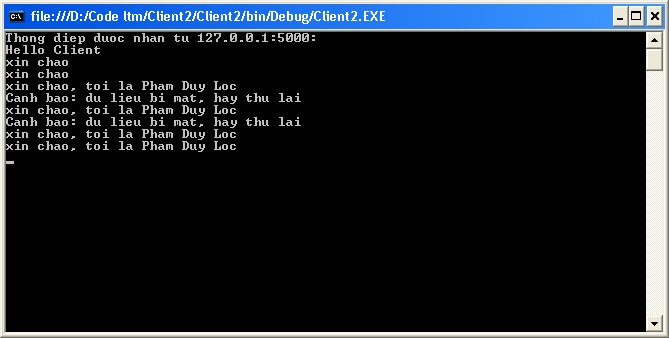
}

#### III.6.3. Test chương trình

Chạy chương trình server:



Chạy chương trình client:



#### III.6.4. Trả lời câu hỏi:

2) Khi dùng phương pháp này các dữ liệu bị mất do kích thước bộ đệm ban đầu nhỏ có lấy lại được không?

Không, các dữ liệu đã mất sẽ không lấy lại được.

- Chương trình hiện tại chỉ được thiết kế để báo lỗi về mất kết nối hay không nhận được gói tin trong một khoảng thơi gian nhất định

- Việc nhận một tập tin lơn hơn kích thước bộ nhớ đệm ban đầu không gây ra lỗi 🡪 Chương trình sẽ tiếp tục chạy mà không biết dữ liệu bị mất. Việc tăng kích thước bộ nhớ đệm chỉ có tác dụng với những lần nhận sau 🡪 không thể khôi phục được phần dữ liệu đã bị mất nếu không có một cơ chế yêu cầu gửi lại dữ liệu.

## III.7. Ngăn cản mất gói tin khi lập trình mạng sử dụng giao thức UDP

#### III.7.1. Hướng dẫn lập trình để ngăn cản mất gói tin

Khi gởi dữ liệu bằng giao thức UDP với các chương trình UDP đơn giản ở trên, nếu dữ liệu không đến được tới đích vì một lý do nào đó thì không thể nào biết được gói tin gởi đi đã bị mất.

Để ngăn cản mất gói tin ta dùng phương thức SetSocketOption() để thiết lập giá trị TimeOut để sau một thời gian không nhận được hồi báo thì gởi lại dữ liệu server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveTimeout, 3000); Và viết lại hàm gởi và nhận dữ liệu:

private int SndRcvData(Socket s, byte[] message, EndPoint rmtdevice)

{ int recv; int retry = 0; while (true)

{

Console.WriteLine("Truyen lai lan thu: #{0}", retry); try

{

s.SendTo(message, message.Length, SocketFlags.None, rmtdevice); data = new byte[1024];

recv = s.ReceiveFrom(data, ref Remote);

}

catch (SocketException)

{ recv = 0;

} if (recv > 0)

{

return recv;

} else { retry++; if (retry > 4)

{ return 0;

}

}

}

}

Viết lại class điều khiển việc gởi và nhận dữ liệu ngăn cản mất gói tin

public RetryUdpClient()

{

string input, stringData; int recv;

IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);

Socket server = new Socket(AddressFamily.InterNetwork,

SocketType.Dgram, ProtocolType.Udp);

int sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,

SocketOptionName.ReceiveTimeout);

Console.WriteLine("Gia tri timeout mac dinh: {0}", sockopt); server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveTimeout, 3000);

sockopt = (int)server.GetSocketOption(SocketOptionLevel.Socket,

SocketOptionName.ReceiveTimeout);

Console.WriteLine("Gia tri timeout moi: {0}", sockopt); string welcome = "Xin chao Server"; data = Encoding.ASCII.GetBytes(welcome); recv = SndRcvData(server, data, ipep); if (recv > 0)

{

stringData = Encoding.ASCII.GetString(data, 0, recv);

Console.WriteLine(stringData);

} else

{

Console.WriteLine("Khong the lien lac voi thiet bi o xa"); return; } while (true)

{

input = Console.ReadLine(); if (input == "exit") break;

recv = SndRcvData(server, Encoding.ASCII.GetBytes(input), ipep); if (recv > 0)

{

stringData = Encoding.ASCII.GetString(data, 0, recv);

Console.WriteLine(stringData);

} else

Console.WriteLine("Khong nhan duoc cau tra loi");

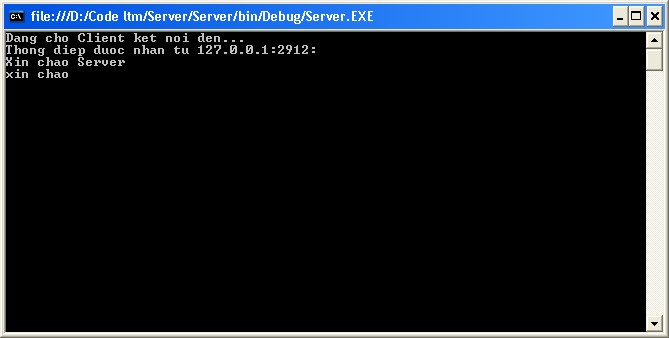
}

Console.WriteLine("Dang dong client"); server.Close();

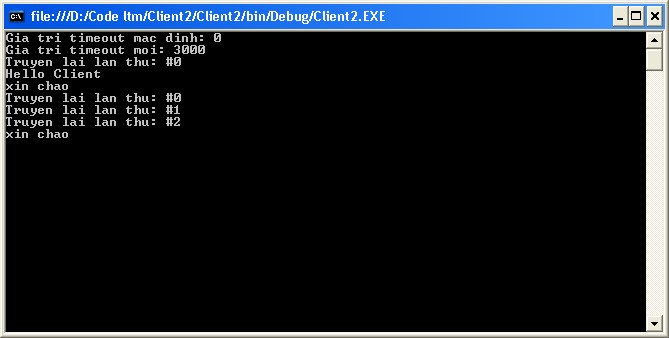
}

#### III.7.2. Test chương trình

Để test chương trình ta dùng hàm Sleep() trong vòng lặp While ở server để giả lập lỗi Chạy chương trình server:



Chạy chương trình client:



#### III.7.3. Trả lời câu hỏi:

3) Sử dụng phương pháp này có ngăn cản triệt để mất dữ liệu không? Nếu không hãy đề xuất phương pháp khác

Không phương pháp này không ngăn mất dữ liệu triệt để vì phương pháp này còn có những vấn đề như:

- Mất gói tin phản hồi (ACK): gói tin mà server gửi để phản hồi cho client có thể mất trên đường truyền 🡪 client gửi lại gói tin khiến server phải xử lý một dữ liệu hai lần

- Giới hạn số lần gửi lại: chương trình hiện chỉ thử gửi lại một số lần nhất định 🡪 nếu mạng bị tắc nghẽn và tất cả lần gửi đều thất bại dữ liệu cuối cùng sẽ bị mất.

- Mất dữ liệu do bộ nhớ đệm: chương trình chưa giải quyết mất 1 phần gói tin do bộ nhớ đệm quá nhỏ 🡪 nếu server gửi 1 gói tin lớn hơn bộ nhớ đệm của client dữ liệu vẫn bị mất một phần.

Các phương pháp khác:

- Đánh số thứ tự cho gói tin (Sequence Numbers): giúp phát hiện trùng lặp và sắp xếp lại gói tin.

- Kiểm soát nguồn (Flow Control) và kiểm soát tắc nghẽn (Congestion Control):

+ Cửa sổ trượt (Sliding Window): phía nhận thông báo cho phía gửi kích thước bộ nhớ đệm còn trống 🡪 tránh việc gửi dữ liệu hơn mức bộ nhớ đệm có thể xử lý.

+ Kiểm soát tắc nghẽn: điều chỉnh tốc độ gửi dữ liệu theo tình trạng mạng, tránh gấy tắc nghẽn và hạn chế mất gói tin.

- Gói tin xác nhận có chọn lọc (Selective Acknowledgment - SACK):

+ Server gửi thông tin về tất cả các gói tin nhận được trong một khoảng thời gian 🡪 client biết chính xác các gói tin nào bị mất và gửi lại gói đã mất thay gửi lại toàn bộ.