**MSSV: 2312647**

**Họ tên: Lê Anh Khoa**

**Lớp: CTK47A**

**LAB 3**

[I. Mục đích 3](#_Toc22764)

[II. Yêu cầu 3](#_Toc22765)

[III. Hướng dẫn 3](#_Toc22766)

[III.1. Xây dựng chương trình UDP Client – Server đơn giản 3](#_Toc22767)

[III.1.1. Hướng dẫn lập trình UDP Server 3](#_Toc22768)

[III.1.2. Hướng dẫn lập trình UDP Client 4](#_Toc22769)

[III.1.3. Test chương trình: 4](#_Toc22770)

[III.1.4. Trả lời câu hỏi: 5](#_Toc22771)

[III.2. Bài tập 5](#_Toc22772)

[III.3. Cải tiến chương trình UDP client-server để có thể gởi và nhận dữ liệu liên tục 6](#_Toc22773)

[III.3.1. Hướng dẫn lập trình UDP Client 6](#_Toc22774)

[III.3.2. Hướng dẫn lập trình UDP Server 6](#_Toc22775)

[III.3.3. Test chương trình 6](#_Toc22776)

[III.3.4. Trả lời câu hỏi: 7](#_Toc22777)

[III.4. Sử dụng phương thức Connect ở client để thiết lập kết nối trước với server 8](#_Toc22778)

[III.4.1. Hướng dẫn lập trình 8](#_Toc22779)

[III.4.2. Test chương trình 8](#_Toc22780)

[III.4.3. Trả lời câu hỏi: 9](#_Toc22781)

[III.5. Kiểm tra khả năng phân biệt biên thông điệp của giao thức UDP 10](#_Toc22782)

[III.5.1. Hướng dẫn lập trình UDP Client 10](#_Toc22783)

[III.5.2. Hướng dẫn lập trình UDP Client 10](#_Toc22784)

[III.5.3. Test chương trình 10](#_Toc22785)

[III.5.4. Trả lời câu hỏi: 11](#_Toc22786)

[III.6. Ngăn cản mất dữ liệu khi lập trình mạng sử dụng giao thức UDP 12](#_Toc22787)

[III.6.1. Trả lời câu hỏi: 12](#_Toc22788)

[III.6.2. Hướng dẫn lập trình để ngăn cản mất dữ liệu 12](#_Toc22789)

[III.6.3. Test chương trình 13](#_Toc22790)

[III.6.4. Trả lời câu hỏi: 14](#_Toc22791)

[III.7. Ngăn cản mất gói tin khi lập trình mạng sử dụng giao thức UDP 14](#_Toc22792)

[III.7.1. Hướng dẫn lập trình để ngăn cản mất gói tin 14](#_Toc22793)

[III.7.2. Test chương trình 17](#_Toc22794)

[III.7.3. Trả lời câu hỏi: 18](#_Toc22795)

# 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

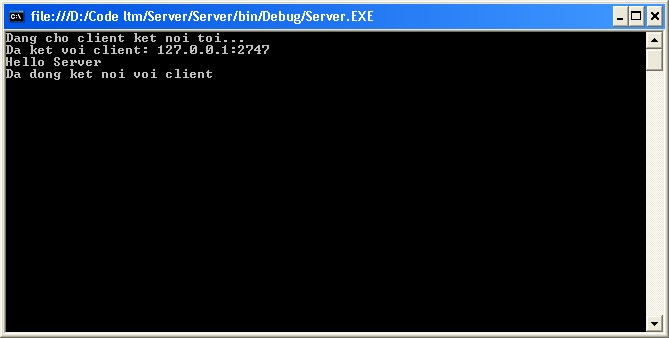
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

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

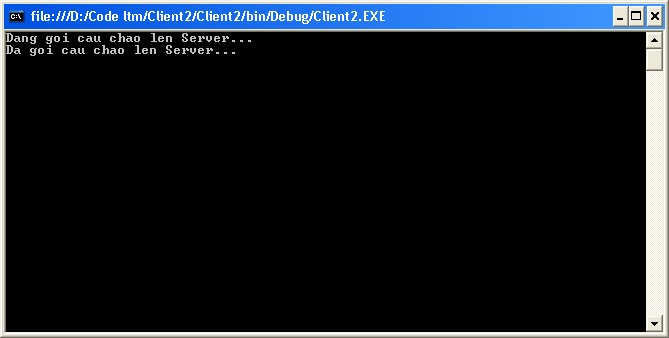
server.SendTo(buff, buff.Length, SocketFlags.None, serverEndPoint);

##### 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 ephemeral port (port ngẫu nhiên) do hệ điều hành cấp cho socket client khi nó gửi dữ liệu đi. Server chỉ cố định port (vd 5000), còn client thì hệ điều hành chọn port rảnh.

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

* Không. Mỗi lần chạy client, hệ điều hành có thể chọn một port khác nhau.

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

* Vì UDP là không kết nối, server không biết ai sẽ gửi dữ liệu đến. Client phải gửi một gói trước để server biết địa chỉ (IP + port) của client để trả lời.

### 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);

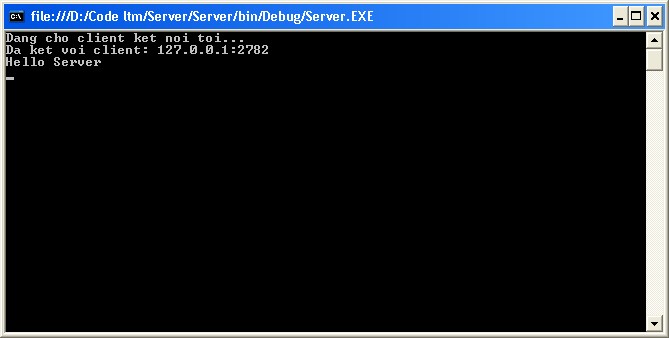
str = Encoding.ASCII.GetString(buff, 0, byteReceive); Console.WriteLine(str);

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

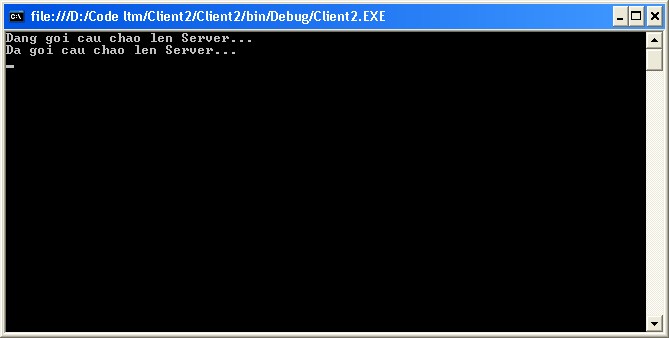
}

##### 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 ?**

Lỗi xảy ra vì đoạn code client:

* Gửi dữ liệu lên server.
* Ngay lập tức chờ nhận phản hồi từ server bằng phương thức ReceiveFrom().

Tuy nhiên, server nhận được dữ liệu, in ra màn hình nhưng lại không gửi phản hồi ngay lập tức cho client. Điều này có thể khiến client bị tắc nghẽn hoặc gặp lỗi SocketException (timeout) khi cố gắng nhận dữ liệu mà chưa có dữ liệu nào được gửi đến từ server.

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

Khi server chưa bật, chương trình client sẽ bị lỗi khi cố gắng gửi dữ liệu. Mặc dù giao thức UDP là không có kết nối, nhưng khi client gửi gói tin đến một địa chỉ mà không có server lắng nghe, có thể xảy ra lỗi SocketException vì không thể tìm thấy đích đến.

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

Khi client tắt, chương trình server không bị lỗi. Server vẫn tiếp tục chạy trong vòng lặp vô hạn và chờ nhận dữ liệu từ bất kỳ client nào khác. Server không duy trì một kết nối liên tục với client, nên việc client đóng không ảnh hưởng đến hoạt động của server.

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

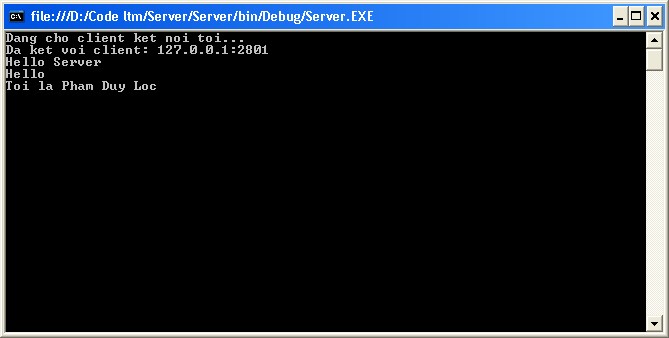
##### III.4.1. Hướng dẫn lập trình

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

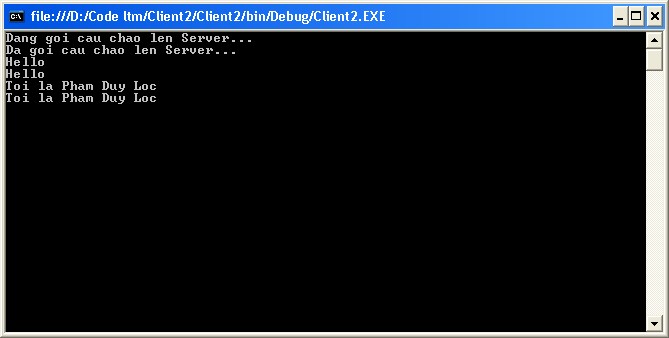
Sau khi gọi phương thức Connect() xong, ta có thể lập trình giống như lập trình với giao thức UDP với client

##### 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?**

Khi chạy chương trình client mà server chưa được bật, client sẽ bị lỗi ngay lập tức. Hiện tượng này xảy ra bởi vì phương thức Connect() mà chient sử dụng để thiết lập kết nối logic với server. Mặc dù UDP là giao thức không có kết nối, việc gọi Connect() sẽ yêu cầu hệ điều hành xác minh rằng một điểm cuối (endpoint) đang lắng nghe ở địa chỉ đó. Nếu server chưa được khởi động, kết nối sẽ thất bại, và chương trình client sẽ ném ra một exception vì không tìm thấy đích đến để kết nối.

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

Khi server bị tắt trong lúc chương trình client đang chạy và sau khi client đã thiết lập kết nối logic:

* Chương trình client sẽ bị lỗi (ném ra SocketException) khi cố gắng thực hiện lệnh client.Receive(recvBuf) tiếp theo.

Vì:

* Ở đây, client đã sử dụng client.Connect(remote) để thiết lập một kết nối logic với server. Mặc dù UDP là giao thức không trạng thái (stateless) và không kết nối, việc gọi Connect() sẽ khóa socket của client vào địa chỉ IP và port cụ thể của server.
* Khi server tắt, hệ điều hành của server sẽ gửi một gói tin ICMP (Internet Control Message Protocol) loại Port Unreachable (Không thể truy cập cổng) về cho client để báo rằng cổng đích đã đóng.
* Khi client cố gắng nhận phản hồi bằng client.Receive(recvBuf) cho dữ liệu đã gửi trước đó, hệ điều hành của client nhận được thông báo ICMP này, nhận ra kết nối logic đã bị hỏng, và ngay lập tức ném ra ngoại lệ SocketException để báo lỗi cho ứng dụng client.

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

Khi client bị tắt chương trình server không bị lỗi và vẫn tiếp tục chạy bình thường.

Vì:

* Giao thức UDP là không có kết nối (connectionless) và server của bạn được thiết kế để luôn lắng nghe trong vòng lặp vô hạn (while(true)).
* Server sử dụng serverSocket.ReceiveFrom(..., ref remote) để chờ nhận bất kỳ gói tin nào từ bất kỳ địa chỉ nào.
* Khi client tắt, nó chỉ đơn giản là đóng socket cục bộ của mình (client.Close()). Hành động này không gửi bất kỳ thông báo nào đến server. Server không duy trì một danh sách các kết nối đang hoạt động (như TCP), do đó việc một client ngừng gửi dữ liệu không ảnh hưởng đến vòng lặp chờ nhận của server. Server vẫn sẵn sàng để nhận dữ liệu 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);

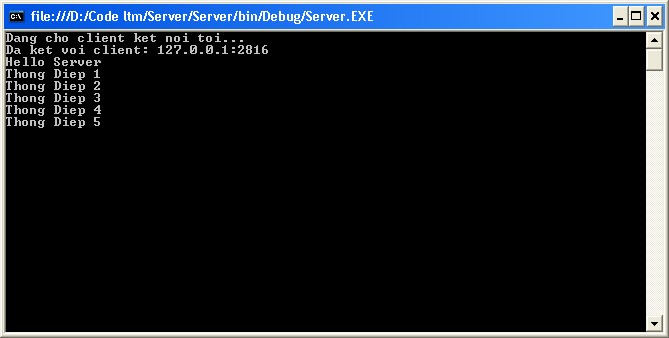
str = Encoding.ASCII.GetString(buff, 0, byteReceive);

Console.WriteLine(str);

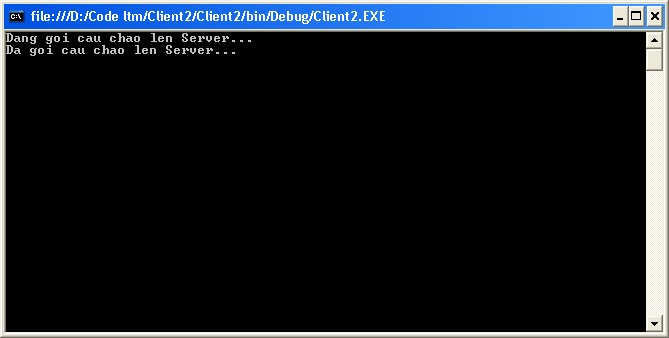
}

##### 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 ?**

## Khi lập trình bằng giao thức UDP, các thông điệp được phân biệt với nhau vì UDP là một giao thức dựa trên gói tin (datagram-based). Mỗi lần client gọi SendTo() để gửi dữ liệu, nó sẽ đóng gói toàn bộ dữ liệu đó thành một gói tin UDP độc lập. Gói tin này chứa thông tin đầy đủ về dữ liệu, địa chỉ nguồn và địa chỉ đích, và được gửi đi một cách riêng biệt.

Khi server nhận được một gói tin, nó sẽ đọc toàn bộ dữ liệu của gói đó một lần duy nhất bằng phương thức ReceiveFrom(). Do đó, mỗi lần ReceiveFrom() được gọi, server sẽ nhận được một gói tin hoàn chỉnh và riêng biệt. Điều này cho phép server phân biệt rõ ràng từng thông điệp được gửi từ client, ngay cả khi chúng được gửi liên tiếp nhau.

Trong bài, vòng lặp for ở client gửi 5 thông điệp riêng biệt. Vòng lặp for tương ứng ở server sẽ nhận và xử lý 5 gói tin đó một cách tuần tự, giúp chúng hiển thị riêng rẽ và phân biệt trên màn hình.

## 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?**

#### Có, sẽ xảy ra lỗi mất dữ liệu. Điều này xảy ra vì sau khi gửi dữ liệu, đã khởi tạo bộ đệm (buff – new byte[10]) với kích thước cố định là 10 byte. Khi server gửi phản hồi lại và kích thước của phản hồi lớn hơn 10 byte, phương thức ReceiveFrom() sẽ chỉ đọc 10 byte đầu tiên, các byte còn lại sẽ bị cắt bỏ dẫn đến mất mát dữ liệu.

#### 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)

{

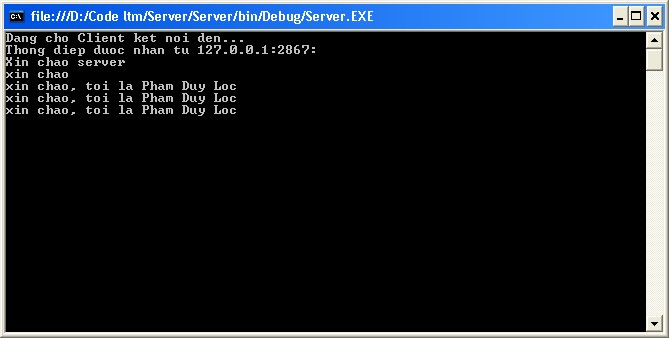
Console.WriteLine("Canh bao: du lieu bi mat, hay thu lai"); i += 10;

}

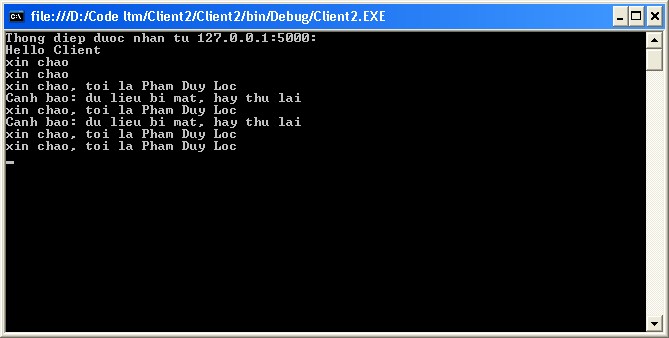
}

#### 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, dữ liệu đã bị mất do kích thước bộ đệm ban đầu nhỏ **không thể lấy lại được** bằng phương pháp này.

Vì:

* Đoạn code bạn cung cấp chỉ xử lý lỗi khi nhận dữ liệu. Cụ thể, khi một SocketException xảy ra do dữ liệu nhận được lớn hơn kích thước bộ đệm (new byte[i]), chương trình sẽ in ra một cảnh báo và tăng kích thước của bộ đệm (i += 10) cho lần nhận tiếp theo.
* Tuy nhiên, phương pháp này không có cơ chế nào để yêu cầu server gửi lại gói tin đã bị mất. Khi dữ liệu bị tràn bộ đệm, nó sẽ bị cắt bỏ và không thể phục hồi. Chương trình chỉ có thể hy vọng rằng các gói tin tiếp theo sẽ được nhận toàn vẹn với kích thước bộ đệm mới.

## 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");

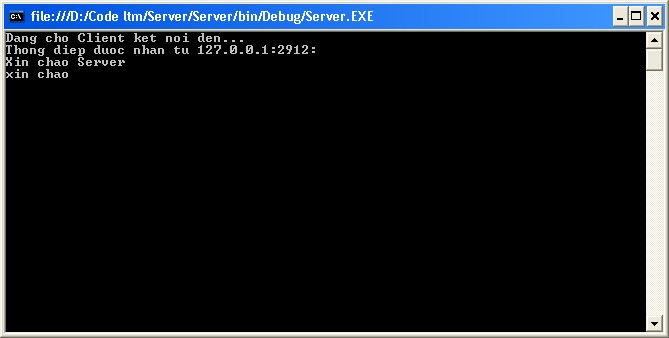
}

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

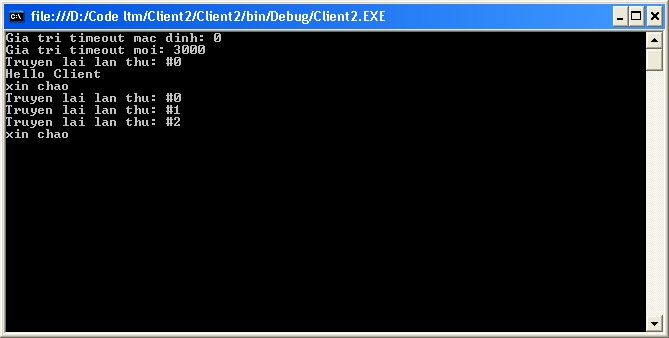
}

#### 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**

Phương pháp không thể ngăn chặn triệt để việc mất gói tin.

Phương pháp sử dụng SetSocketOption() để đặt ReceiveTimeout và viết lại hàm gửi/nhận (SndRcvData) chỉ giúp giảm thiểu việc mất gói tin chứ không thể loại bỏ hoàn toàn.

* Không triệt để vì: Giao thức UDP bản thân nó không có cơ chế đảm bảo gói tin được gửi đến đích. Khi một gói tin bị mất trên đường truyền, server không hề biết đến sự tồn tại của nó, và client cũng chỉ biết được điều đó sau khi hết thời gian chờ (ReceiveTimeout). Vòng lặp gửi lại chỉ là một cách để thử lại, nhưng nếu tình trạng tắc nghẽn mạng diễn ra liên tục, gói tin vẫn có thể tiếp tục bị mất.
* Tầm quan trọng của ACK: Trong lập trình mạng, để đảm bảo độ tin cậy, cần có một cơ chế xác nhận đã nhận (ACK - acknowledgment). UDP không có cơ chế này.

Đề xuất phương pháp khác:

Để ngăn chặn việc mất dữ liệu một cách triệt để hơn, nên xây dựng một giao thức độ tin cậy tùy chỉnh trên nền UDP, bao gồm các thành phần sau:

* Đánh số thứ tự gói tin (Sequence Number): Client sẽ gán một số thứ tự duy nhất cho mỗi gói tin gửi đi.
* Xác nhận đã nhận (ACK): Khi server nhận được một gói tin, nó phải gửi lại một gói tin xác nhận (ACK) về cho client, kèm theo số thứ tự của gói tin đã nhận.
* Thời gian chờ (Timeout) và gửi lại: Client sau khi gửi gói tin sẽ khởi động một bộ đếm thời gian. Nếu không nhận được ACK tương ứng trong khoảng thời gian này, client sẽ tự động gửi lại gói tin đó. Quá trình này sẽ lặp lại cho đến khi nhận được ACK hoặc đạt đến giới hạn số lần thử lại.