**LAB 2**

# I. Mục đích

Ban đầu làm quen với lập trình Socket

Hiểu được cách hoạt động của bộ đệm hệ thống và bộ đệm chương trình

Xây dựng chương trình client-server đơn giản

**II. Yêu cầu**

Xây dựng các chương trình client-server từ mức đơn giản nhất rồi phức tạp dần

# III. Hướng dẫn

## III.1. Tạo một chương trình server đơn giản

### III.1.1. Hướng dẫn lập trình

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.Any, 5000);

Tạo Server Socket, Socket này sẽ được kết nối với Server EndPoint:

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

ProtocolType.Tcp);

Lắng nghe kết nối trên Server Socket, khi có client kết nối tới sẽ trả về Socket để trao đổi dữ liệu với client:

serverSocket.Listen(10);

Socket clientSocket = serverSocket.Accept();

Khi client kết nối tới trên server sẽ xuất thông tin của client kết nối tới (địa chỉ IP, Port)

EndPoint clientEndPoint = clientSocket.RemoteEndPoint; Console.WriteLine(clientEndPoint.ToString());

**III.1.2. Test chương trình:**

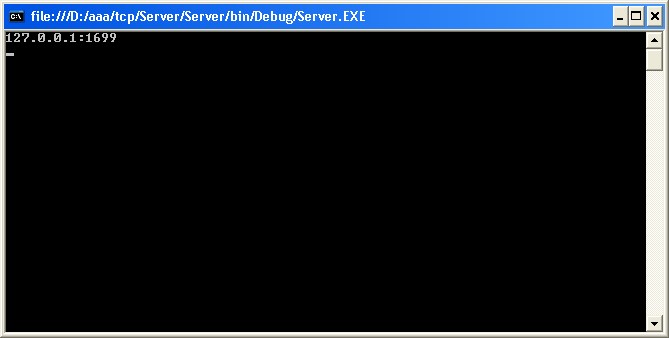
Chạy chương trình Server:



Chạy chương trình CMD và telnet vào server trên với địa chỉ IP loopback và port 5000:



Kết quả trả về trên Server:



**III.1.3. Trả lời câu hỏi:**

1. Khi telnet vào Server bằng port 5000 tại sao port client dùng để kết nối vào lại không phải là port 5000 ?

Khi kết nối TCP được tạo cần 4 cặp giá trị để xác đinh duy nhất (IP nguồn, port nguồn, IP đích, port đích)

- Server: Lắng nghe trên 1 cổng cố định (5000) để client biết nơi để kết nối (port đích).

- Client: khi tạo kết nối, hệ điều hành tự động gán cho một cổng tạm thời đây là cổng nguồn.

🡪 Client không thể dụng cổng 5000 vì server đang dùng, client cần cổng riêng để giử và nhận dữ liệu.

1. Như hình trên, giải thích vì sao có client dùng port giá trị 1699 ?

Số cổng 1699 là một cổng tạm thời (ephemeral port) được hệ điều hành của máy client tự động chọn và gán cho ứng dụng telnet tại thời điểm kết nối.

1. Có phải lúc nào client cũng mở port 1699 để kết nối với Server không ?

Không, cổng này được hệ điều hành cấp phát động và sẽ khác nhau trong mỗi lần kết nối mới. Nếu bạn đóng cửa sổ telnet và kết nối lại, rất có thể bạn sẽ thấy một số cổng khác.

1. Ta có thể chạy 2 chương trình CMD để telnet vào Server trên được không ? Giải thích.

Không, với chương trình này không thể chạy 2 chương trình CMD cùng lúc vì: Chương trình server hiện tại chỉ là một server đơn luồng (single-threaded). Sau khi gọi serverSocket.Accept(), nó sẽ đợi một client kết nối. Khi client đầu tiên kết nối, phương thức Accept() trả về clientSocket và chương trình tiếp tục thực thi các dòng lệnh tiếp theo. Tuy nhiên, nó không quay lại vòng lặp để gọi Accept() một lần nữa. Do đó, nó chỉ có thể xử lý một kết nối tại một thời điểm. Mọi client khác cố gắng kết nối sẽ phải đợi trong hàng đợi (backlog, được định nghĩa là 10 trong serverSocket.Listen(10)) cho đến khi server sẵn sàng chấp nhận kết nối mới, điều mà chương trình này sẽ không bao giờ làm. Để xử lý nhiều client cùng lúc, bạn cần sử dụng đa luồng (multi-threading) hoặc các kỹ thuật lập trình bất đồng bộ.

## III.2. Tạo chương trình Server đơn giản, khi Client kết nối vào thì gởi câu chào cho client

### III.2.1. Hướng dẫn lập trình

Sử dụng chương trình Server đơn giản ở trên, thêm phần gởi dữ liệu xuống client

byte[] buff;

string hello = "Hello Client"; buff = Encoding.ASCII.GetBytes(hello); clientSocket.Send(buff, 0, buff.Length, SocketFlags.None);

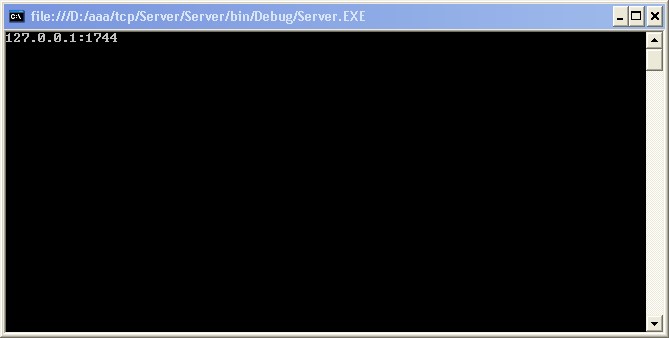
**III.2.2. Test chương trình** Chạy chương trình Server:



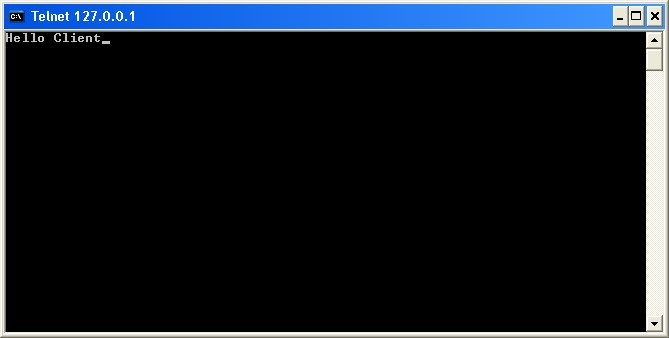
Chạy chương trình CMD và telnet vào server trên với địa chỉ IP loopback và port 5000:



Kết quả trả về trên Server:



Kết quả trả về trên client :



**III.2.3. Trả lời câu hỏi:**

1. Tạo sao trong đoạn code viết thêm vào ta không cần phải khởi tạo độ lớn ban đầu cho buffer ?

Vì phương thức Encoding.ASCII.GetBytes() sẽ tự động thực hiện việc này:

- Tính tians chính xác cần bào nhiêu byte để hiển thị chuỗi được truyền vào

- Điền dữ liệu đã mã hóa vào mảng buff.

- Trả về mảng đã được tạo và cấp phát bộ nhớ.

🡪 Mảng có kích thước phù hợp để biểu diễn chuỗi tránh gửi các byte rác và tiết kiệm bộ nhớ.

1. Giải thích ý nghĩa các tham số của phương thức clientSocket.Send .

clientSocket.Send(buff, 0, buff.Length, SocketFlags.None);

- buff: mảng chứa dữ liệu muốn gửi đi.

- 0: tham số offset, chỉ định vị trí ban đầu trong mảng để gửi đi (bắt đầu gửi dữ liệu từ đầu bảng).

- buff.Lenght: độ dài mảng, chỉ định số lượng byte sẽ gửi đi tính từ vị trí offset 🡪 gửi taonf bộ nội dung mảng.

- SocketFlags.None: không thêm tùy chọn đặc biệt, chỉ gửi theo mặc định.

## III.3. Xây dựng chương trình client đơn giản

### III.3.1. Hướng dẫn lập trình

Để lập trình socket ta using 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.Loopback, 5000);

Tạo socket dùng để kết nối với server:

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

ProtocolType.Tcp);

Console.WriteLine("Dang ket noi voi server..."); serverSocket.Connect(serverEndPoint);

Nếu kết nối thành công, nhận câu chào từ server và hiển thị câu chào lên màn hình:

if (serverSocket.Connected)

{

Console.WriteLine("Ket noi thanh cong voi server ...");

byteReceive = serverSocket.Receive(buff, 0, buff.Length, SocketFlags.None);

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

Console.WriteLine(str);

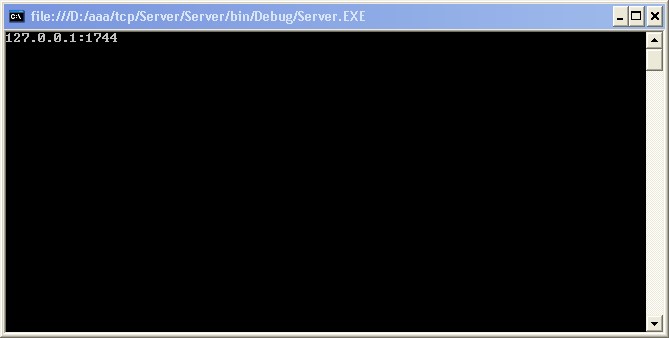
}

**III.3.2. Test chương trình**

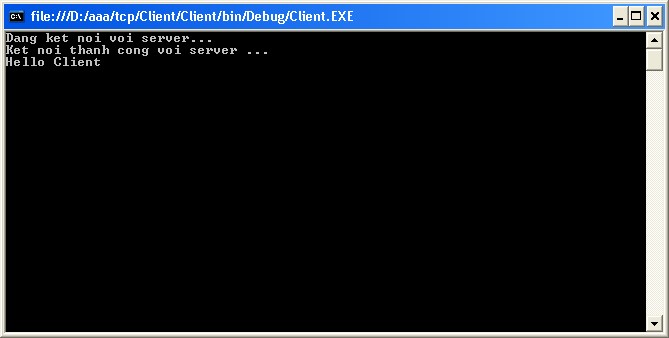
Chạy chương trình Server:



Kết quả trả về trên Server:



Kết quả trả về trên client :



**III.3.3. Trả lời câu hỏi:**

1. Khi client kết nối đến server mà server chưa chạy hoạc là đường truyền mạng bị rớt thì có hiện tượng gì xảy ra ?. Giải thích ?

- Hiện tượng: Chương trình client sẽ bị dừng ở dòng serverSocket.Connect(serverEndPoint); trong một khoảng thời gian (thường là vài giây), sau đó chương trình sẽ bị lỗi và dừng lại (crash) với một ngoại lệ System.Net.Sockets.SocketException. Thông báo lỗi thường sẽ là "No connection could be made because the target machine actively refused it" (Không thể kết nối vì máy tính đích đã từ chối kết nối).

- Giải thích: Khi client cố gắng kết nối, nó sẽ gửi một gói tin (packet) đến địa chỉ IP và cổng của server. Nếu server không chạy, hoặc có tường lửa chặn, hoặc đường truyền mạng bị ngắt, gói tin này sẽ không nhận được phản hồi. Sau một khoảng thời gian chờ (timeout), hệ điều hành sẽ xác định rằng kết nối không thể được thiết lập và báo lỗi cho chương trình client thông qua một SocketException.

1. Từ đoạn code ở chương trình client: str = Encoding.ASCII.GetString(buff, 0, byteReceive)

Nếu ta thay byteReceive bằng buff.Lengh thì kết quả sẽ như thế nào? Giải thích.

- Kết quả: Thay vì chỉ hiển thị chuỗi "Hello Client", chương trình sẽ hiển thị thêm rất nhiều ký tự rác hoặc khoảng trống phía sau. Ví dụ: "Hello Client\0\0\0\0\0...".

- Giải thích:

buff là một mảng byte được khởi tạo với kích thước cố định, ví dụ là 1024 byte (new byte[1024]).

byteReceive là số lượng byte thực tế mà server gửi và client nhận được. Ví dụ, chuỗi "Hello Client" chỉ có 11 byte.

Nếu dùng Encoding.ASCII.GetString(buff, 0, buff.Length), tức là đang yêu cầu chương trình chuyển đổi toàn bộ 1024 byte trong bộ đệm thành một chuỗi. Vì chỉ có 11 byte đầu tiên chứa dữ liệu có ý nghĩa, 1013 byte còn lại sẽ là các giá trị mặc định (thường là số 0, tương ứng với ký tự null \0).

Do đó, việc sử dụng byteReceive là rất quan trọng để chỉ xử lý đúng phần dữ liệu đã được gửi, tránh đọc phải dữ liệu rác hoặc các giá trị mặc định còn lại trong bộ đệm.

## III.4. Cải tiến chương trình client

### III.4.1. Hướng dẫn lập trình

Chương trình client này khi kết nối đến server mà server chưa bật hoạc đường truyền bị lỗi thì chương trình sẽ bị lỗi.

Sửa lại chương trình bằng cách thêm try-catch vào:

try {

serverSocket.Connect(serverEndPoint);

}

catch (SocketException se)

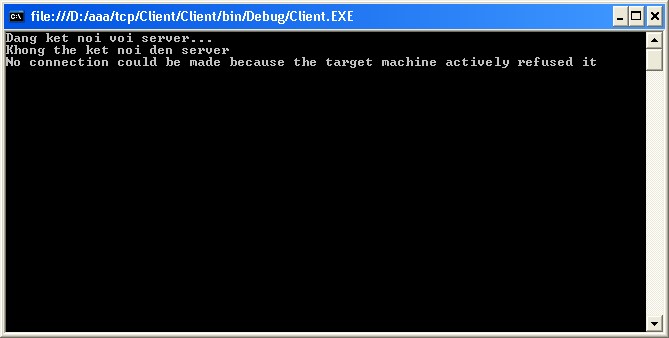
{

Console.WriteLine("Khong the ket noi den server"); return;

}

### III.4.2. Test chương trình

Chạy chương trình ở client mà không chạy server:



## III.5. Thêm khả năng gởi dữ liệu lên server và nhận lại dữ liệu từ server gởi về

### III.5.1. Hướng dẫn lập trình

Việc này được thực hiện bằng cách thêm vòng lặp vô hạn

while (true)

{

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

serverSocket.Send(buff, 0, buff.Length, SocketFlags.None); buff = new byte[1024];

byteReceive = serverSocket.Receive(buff, 0, buff.Length, SocketFlags.None);

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

Console.WriteLine(str); }

## III.6. Thêm khả năng nhận dữ liệu từ client gởi lên và gởi trả dữ liệu ngược lại client

### III.6.1. Hướng dẫn lập trình

Việc này được thực hiện bằng cách thêm vòng lặp vô hạn

while (true)

{

buff = new byte[1024];

byteReceive = clientSocket.Receive(buff, 0, buff.Length, SocketFlags.None);

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

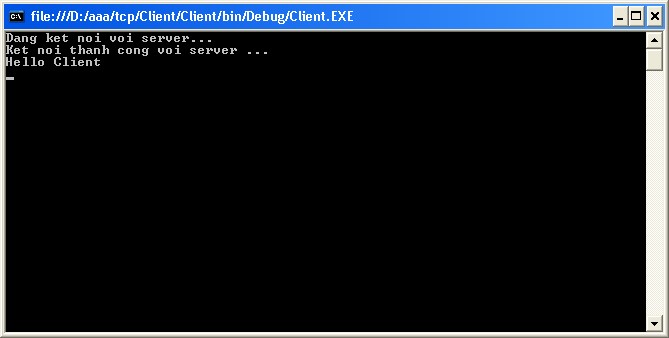
clientSocket.Send(buff, 0, byteReceive, SocketFlags.None); }

### III.6.2. Test chương trình

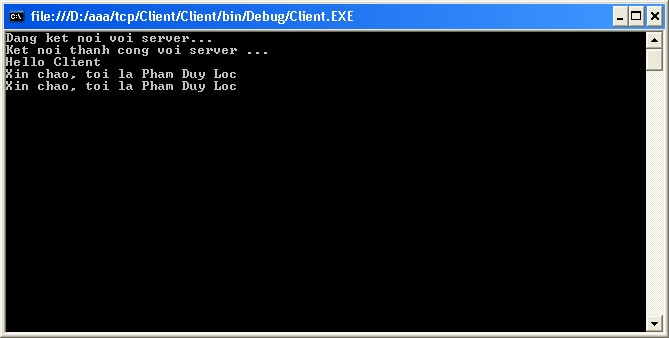
Chạy chương trình server:



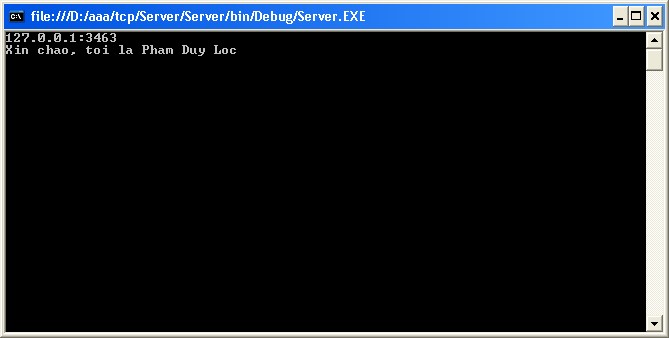
Chạy chương trình client:



Nhập dữ liệu và gởi lên server:



Kết quả hiển thị trên server:



**III.6.3. Trả lời câu hỏi:**

1. Khi thay đổi kích thước bộ đệm từ 1024 xuống 10 thì có hiện tượng gì xảy ra ?. Giải thích

Hiện tượng: Khi client gửi một chuỗi dài hơn 10 ký tự (ví dụ: "Hello, this is a long message"), server sẽ chỉ nhận và phản hồi lại 10 ký tự đầu tiên. Client cũng sẽ chỉ nhận lại 10 ký tự đó từ server. Dữ liệu còn lại của chuỗi sẽ bị "treo" trong luồng (stream) và sẽ được nhận ở lần đọc tiếp theo.

Giải thích:

- TCP là giao thức hướng luồng (Stream-Oriented): TCP đảm bảo dữ liệu được truyền đi một cách tuần tự và không bị mất, nhưng nó không quan tâm đến khái niệm "gói tin" hay "tin nhắn" riêng lẻ. Nó coi dữ liệu như một dòng byte liên tục.

- Socket.Receive(buffer) hoạt động như thế nào: Lệnh này sẽ đọc dữ liệu từ luồng mạng và lấp đầy buffer.

+ Nó sẽ đọc tối đa là bằng kích thước của buffer (trong trường hợp này là 10 byte).

+ Nếu có nhiều hơn 10 byte đang chờ trong luồng, nó sẽ chỉ lấy 10 byte đầu tiên. Phần còn lại vẫn nằm trong hàng đợi của hệ điều hành để được xử lý trong lần gọi Receive() tiếp theo.

+ Nếu có ít hơn 10 byte, nó sẽ lấy hết số byte đang có và trả về.

- Hệ quả: Vì cả client và server đều nằm trong vòng lặp while(true), chúng sẽ liên tục đọc từng "mẩu" 10 byte của dữ liệu cho đến khi không còn gì để đọc trong luồng. Điều này sẽ làm cho việc hiển thị dữ liệu bị ngắt quãng và không đúng với ý định ban đầu là xử lý từng tin nhắn hoàn chỉnh.

1. Nếu ta không reset lại bộ đệm thì có hiện tượng gì xảy ra? Giải thích.

Hiện tượng: Nếu bạn gửi một tin nhắn dài, sau đó gửi một tin nhắn ngắn hơn, tin nhắn ngắn hơn sẽ chỉ ghi đè lên phần đầu của bộ đệm (buffer), và phần còn lại của tin nhắn cũ vẫn còn trong bộ đệm.

Giải thích:

Phương thức Receive() sẽ ghi dữ liệu nhận được vào mảng buff bắt đầu từ vị trí 0, và ghi đè lên byteReceive byte đầu tiên. Nó không tự động xóa các dữ liệu cũ còn lại trong buffer.

Hãy xem xét ví dụ sau nếu chúng ta bỏ dòng buff = new byte[1024]; ra khỏi vòng lặp while:

Lần 1: Client gửi chuỗi "Hello World" (11 ký tự).

Server nhận được 11 byte.

Nội dung của buff sẽ là: ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', 0, 0, ...]

byteReceive là 11.

Encoding.ASCII.GetString(buff, 0, byteReceive) sẽ trả về "Hello World". Mọi thứ vẫn đúng.

Lần 2: Client gửi chuỗi "Hi" (2 ký tự).

Server nhận được 2 byte. Dữ liệu mới này sẽ được ghi vào đầu buff.

Nội dung của buff bây giờ sẽ là: ['H', 'i', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', 0, 0, ...] (chỉ 2 ký tự đầu bị thay đổi).

byteReceive là 2.

Encoding.ASCII.GetString(buff, 0, byteReceive) sẽ trả về "Hi".

Vậy tại sao việc không reset lại là không tốt?

Mặc dù trong ví dụ trên, kết quả in ra vẫn đúng vì đã sử dụng byteReceive để giới hạn độ dài chuỗi cần chuyển đổi. Tuy nhiên, đây là một thói quen lập trình không an toàn. Nếu vì một lý do nào đó mà xử lý toàn bộ buff thay vì chỉ byteReceive byte đầu tiên, bạn sẽ nhận được kết quả sai là "Hillo World...".

Việc khởi tạo lại bộ đệm buff = new byte[1024]; trong mỗi vòng lặp đảm bảo rằng bộ đệm luôn "sạch" trước mỗi lần nhận dữ liệu mới, loại bỏ hoàn toàn nguy cơ xử lý nhầm dữ liệu cũ còn sót lại.

# IV. Bài tập

1. Chương trình trên khi tắt client thì trên server sẽ bị lỗi, hãy cải tiến lại chương để khi client tắt đột ngột trên server sẽ thông báo client đã ngắt kết nối
2. Cải tiến chương trình: khi client nhập chữ “exit” thì sẽ tắt client, trên server thông báo client đã ngắt kết nối
3. Viết chương trình thực hiện các phép tính số học, việc thực hiện tính toán được thực hiện trên server, phía client sẽ nhập vào các toán hạng và các phép tính