

CHƯƠNG 4

Lập trình Socket

Mục đích

Chương này nhằm giới thiệu về cách thức xây dựng ứng dụng Client-Server trên mạng TCP/IP theo cả hai chế độ Có kết nối (TCP) và Không kết nối (UDP).

Yêu cầu

Sau khi hoàn tất chương này, bạn có thể:

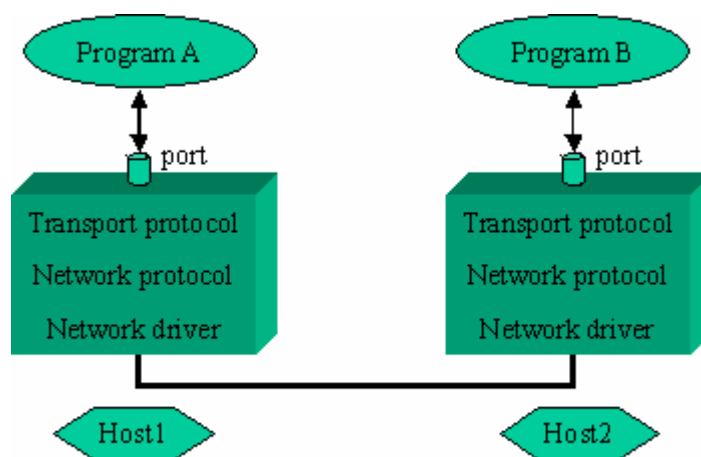
- Giải thích được Socket là gì, vai trò của số hiệu cổng (Port) và địa chỉ IP trong cơ chế Socket.
- Phân biệt được sự khác biệt của hai loại Protocol TCP và UDP.
- Trình bày được các bước xây dựng một chương trình Client-Server sử dụng Socket làm phương tiện giao tiếp trong cả hai chế độ: Có kết nối và không kết nối.
- Liệt kê các lớp hỗ trợ lập trình Socket của Java.
- Xây dựng được các chương trình Client sử dụng Sokcet ở chế độ có kết nối bằng ngôn ngữ Java.
- Xây dựng được các chương trình Server sử dụng Sokcet ở chế độ có kết nối phục vụ tuần tự và phục vụ song song bằng ngôn ngữ Java.
- Xây dựng được các chương trình Client-Server sử dụng Sokcet ở chế độ không kết nối bằng ngôn ngữ Java.
- Tự xây dựng được các Protocol mới cho ứng dụng của mình.

1.1. Giới thiệu về socket

1.1.1. Giới thiệu

Socket là một giao diện lập trình ứng dụng (API-Application Programming Interface). Nó được giới thiệu lần đầu tiên trong ấn bản UNIX - BSD 4.2. dưới dạng các hàm hệ thống theo cú pháp ngôn ngữ C (socket(), bind(), connect(), send(), receive(), read(), write(), close() ...). Ngày nay, Socket được hỗ trợ trong hầu hết các hệ điều hành như MS Windows, Linux và được sử dụng trong nhiều ngôn ngữ lập trình khác nhau: như C, C++, Java, Visual Basic, Visual C++, ...

Socket cho phép thiết lập các kênh giao tiếp mà hai đầu kênh được đánh dấu bởi hai cổng (port). Thông qua các cổng này một quá trình có thể nhận và gửi dữ liệu với các quá trình khác.



Hình 4.1 – Mô hình Socket

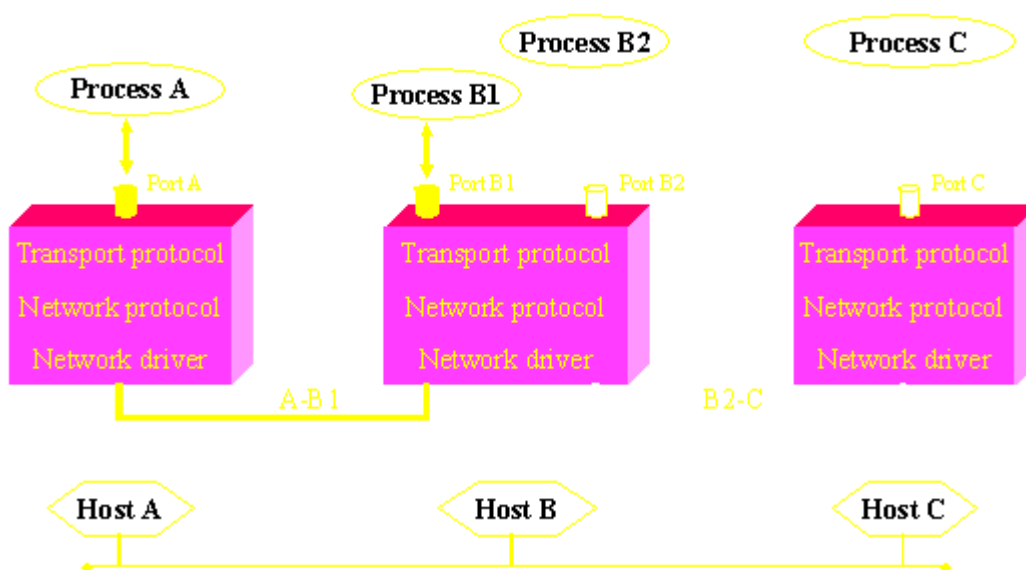
Có hai kiểu socket:

- Socket kiểu AF_UNIX chỉ cho phép giao tiếp giữa các quá trình trong cùng một máy tính
- Socket kiểu AF_INET cho phép giao tiếp giữa các quá trình trên những máy tính khác nhau trên mạng.

1.1.2. Số hiệu cổng (Port Number) của socket

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

Ngoài số hiệu cổng, hai bên giao tiếp còn phải biết địa chỉ IP của nhau. Địa chỉ IP giúp phân biệt máy tính này với máy tính kia trên mạng TCP/IP. Trong khi số hiệu cổng dùng để phân biệt các quá trình khác nhau trên cùng một máy tính.



Hình 4.2 – Cổng trong Socket

Trong hình trên, địa chỉ của quá trình B1 được xác định bằng 2 thông tin: (Host B, Port B1):

Địa chỉ máy tính có thể là địa chỉ IP dạng 203.162.36.149 hay là địa chỉ theo dạng

tên miền như www.cit.ctu.edu.vn

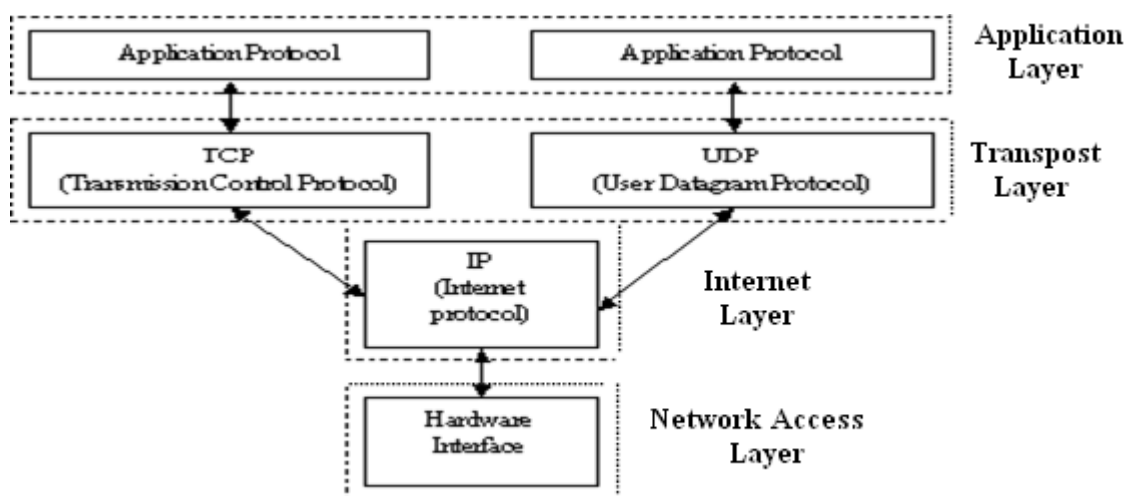
Số hiệu cổng gán cho Socket phải duy nhất trên phạm vi máy tính đó, có giá trị trong khoảng từ 0 đến 65535 (16 bits). Trong đó, các cổng từ 1 đến 1023 được gọi là cổng hệ thống được dành riêng cho các quá trình của hệ thống.

Các cổng mặc định của 1 số dịch vụ mạng thông dụng:

Số hiệu cổng	Quá trình hệ thống
7	Dịch vụ Echo
21	Dịch vụ FTP
23	Dịch vụ Telnet
25	Dịch vụ E-mail (SMTP)
80	Dịch vụ Web (HTTP)
110	Dịch vụ E-mail (POP)

1.1.3. Các chế độ giao tiếp

Xét kiến trúc của hệ thống mạng TCP/IP



Hình 4.3 – Bộ giao thức TCP/IP

Tầng giao vận giúp chuyển tiếp các thông điệp giữa các chương trình ứng dụng với nhau. Nó có thể hoạt động theo hai chế độ:

- Giao tiếp có kết nối, nếu sử dụng giao thức TCP
- Hoặc giao tiếp không kết nối, nếu sử dụng giao thức UDP

Socket là giao diện giữa chương trình ứng dụng với tầng giao vận. Nó cho phép ta chọn giao thức sử dụng ở tầng giao vận là TCP hay UDP cho chương trình ứng dụng của mình.

Bảng sau so sánh sự khác biệt giữa hai chế độ giao tiếp có kết nối và không kết nối:

Chế độ có kết nối (TCP)	Chế độ không kết nối (UDP)
<p>Tồn tại kênh giao tiếp ảo giữa hai bên giao tiếp</p> <p>Dữ liệu được gửi đi theo chế độ bảo đảm: có kiểm tra lỗi, truyền lại gói tin lỗi hay mất, bảo đảm thứ tự đến của các gói tin . . .</p> <p>Dữ liệu chính xác, Tốc độ truyền chậm.</p>	<p>Không tồn tại kênh giao tiếp ảo giữa hai bên giao tiếp</p> <p>Dữ liệu được gửi đi theo chế độ không bảo đảm: Không kiểm tra lỗi, không phát hiện không truyền lại gói tin bị lỗi hay mất, không bảo đảm thứ tự đến của các gói tin . . .</p> <p>Dữ liệu không chính xác, tốc độ truyền nhanh.</p> <p>Thích hợp cho các ứng dụng cần tốc độ, không cần chính xác cao: truyền âm thanh, hình ảnh . . .</p>

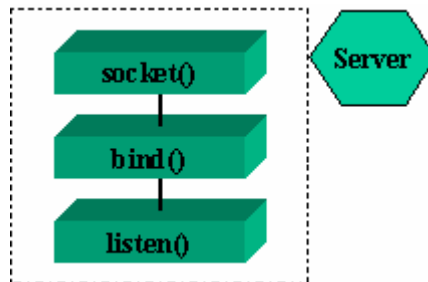
1.2. Xây dựng ứng dụng Client-Server với Socket

Socket là phương tiện hiệu quả để xây dựng các ứng dụng theo kiến trúc Client-Server. Các ứng dụng trên mạng Internet như Web, Email, FTP là các ví dụ điển hình.

Phần này trình bày các bước cơ bản trong việc xây dựng các ứng dụng Client-Server sử dụng Socket làm phương tiện giao tiếp theo cả hai chế độ: Có kết nối và không kết nối.

1.2.1. Mô hình Client-Server sử dụng Socket ở chế độ có kết nối (TCP)

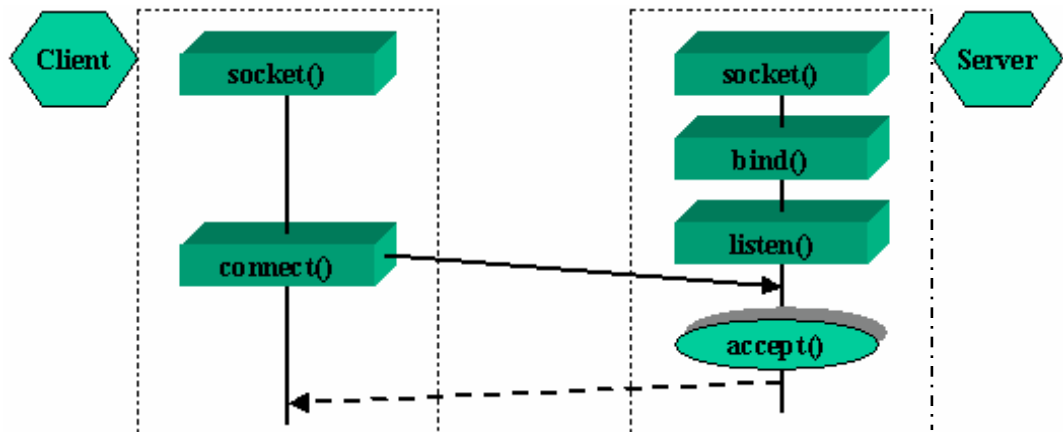
Giai đoạn 1: Server tạo Socket, gán số hiệu cổng và lắng nghe yêu cầu kết nối.



- `socket()`: Server yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng giao vận.
- `bind()`: Server yêu cầu gán số hiệu cổng (port) cho socket.
- `listen()`: Server lắng nghe các yêu cầu kết nối từ các client trên cổng đã được gán.

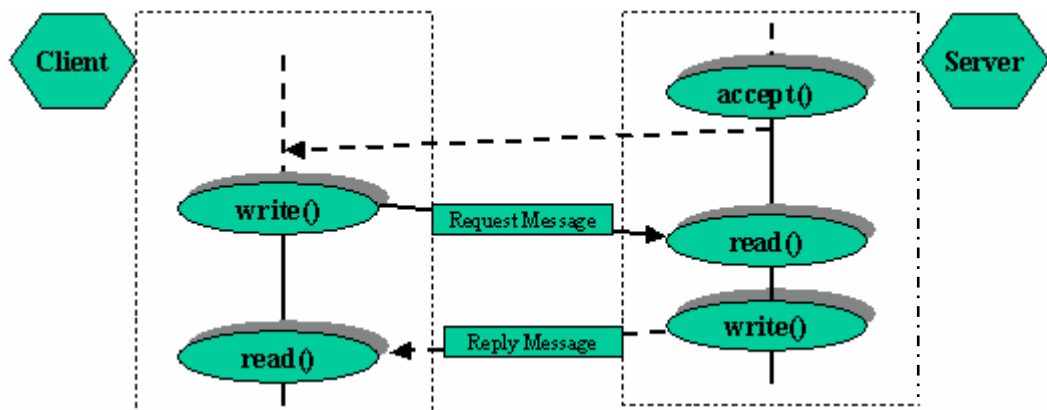
Server sẵn sàng phục vụ Client.

Giai đoạn 2: Client tạo Socket, yêu cầu thiết lập một kết nối với Server.



- `socket()`: Client yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng giao vận, thông thường hệ thống tự động gán một số hiệu cổng còn rảnh cho socket của Client.
- `connect()`: Client gửi yêu cầu kết nối đến server có địa chỉ IP và Port xác định.
- `accept()`: Server chấp nhận kết nối của client, khi đó một kênh giao tiếp ảo được hình thành, Client và server có thể trao đổi thông tin với nhau thông qua kênh ảo này.

Giai đoạn 3: Trao đổi thông tin giữa Client và Server.

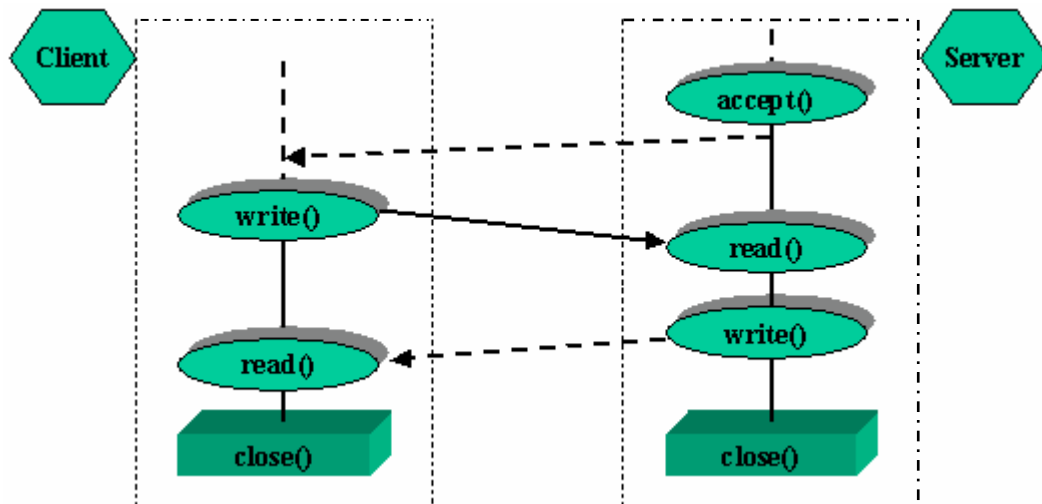


- Sau khi chấp nhận yêu cầu kết nối, thông thường server thực hiện lệnh `read()` và chờ đợi cho đến khi có thông điệp yêu cầu (Request Message) từ client gửi đến.
- Server phân tích và thực thi yêu cầu. Kết quả sẽ được gửi về client bằng lệnh `write()`.
- Sau khi gửi yêu cầu bằng lệnh `write()`, client chờ nhận thông điệp kết quả (ReplyMessage) từ server bằng lệnh `read()`.

Trong giai đoạn này, việc trao đổi thông tin giữa Client và Server phải tuân thủ giao thức của ứng dụng (Dạng thức và ý nghĩa của các thông điệp, qui tắc bắt tay, đồng bộ hóa,...). Thông thường Client sẽ là người gửi yêu cầu đến Server trước.

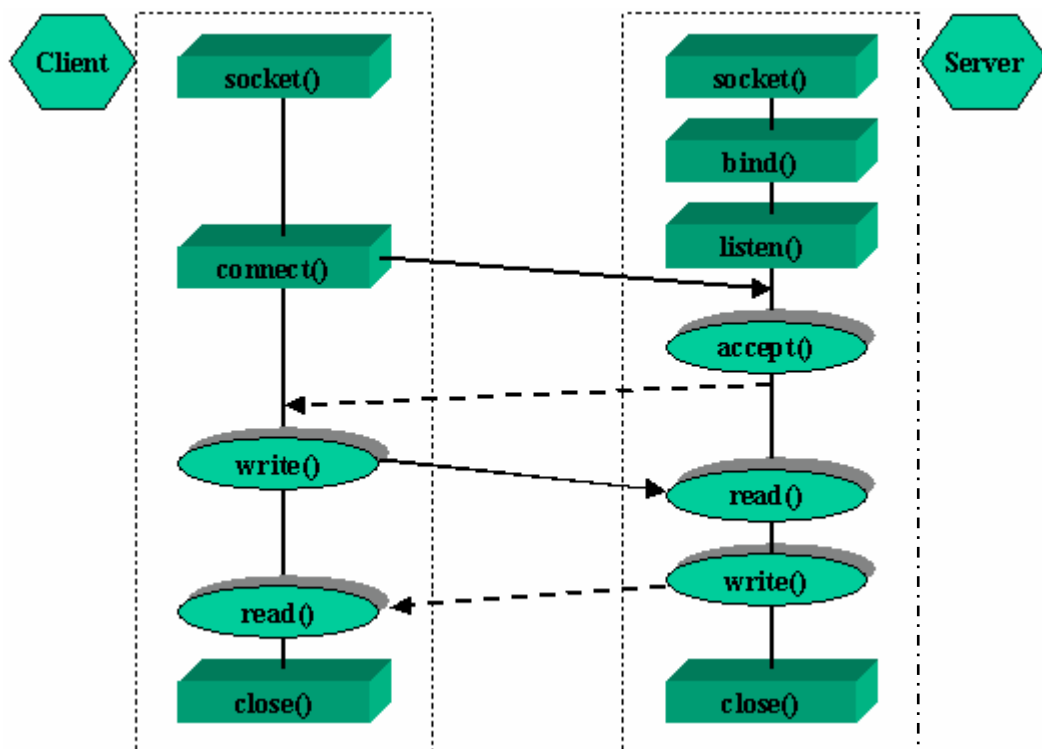
Nếu chúng ta phát triển ứng dụng theo các Protocol đã định nghĩa sẵn, chúng ta phải tham khảo và tuân thủ đúng những qui định của giao thức. Bạn có thể tìm đọc mô tả chi tiết của các Protocol đã được chuẩn hóa trong các tài liệu RFC (Request For Comments). Ngược lại, nếu chúng ta phát triển một ứng dụng Client-Server riêng của mình, thì công việc đầu tiên chúng ta phải thực hiện là đi xây dựng Protocol cho ứng dụng.

Giai đoạn 4: Kết thúc phiên làm việc.



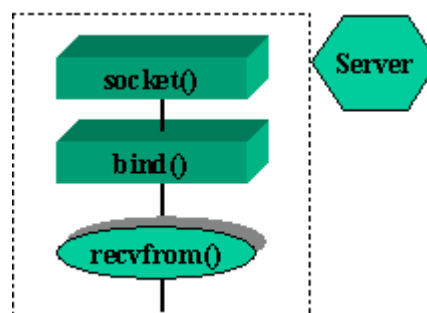
- Các câu lệnh `read()`, `write()` có thể được thực hiện nhiều lần (ký hiệu bằng hình ellipse).
- Kênh ảo sẽ bị xóa khi Server hoặc Client đóng socket bằng lệnh `close()`.

Như vậy toàn bộ tiến trình diễn ra như sau:



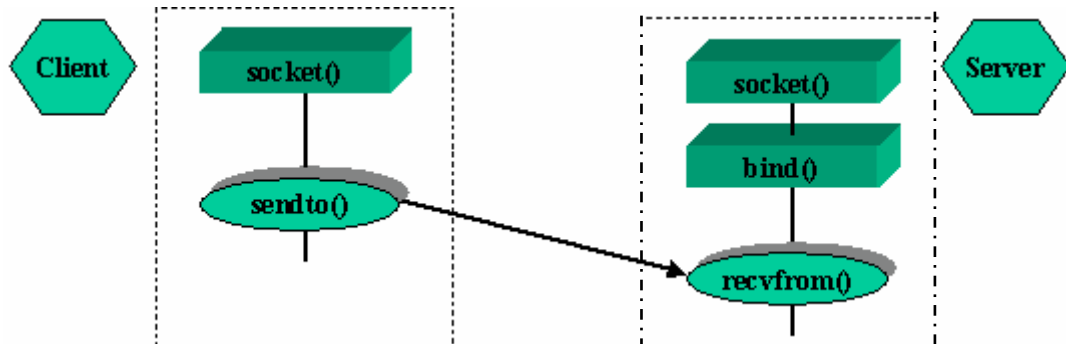
1.2.2. Mô hình Client-Server sử dụng Socket ở chế độ không kết nối (UDP)

Giai đoạn 1: Server tạo Socket - gán số hiệu cổng.

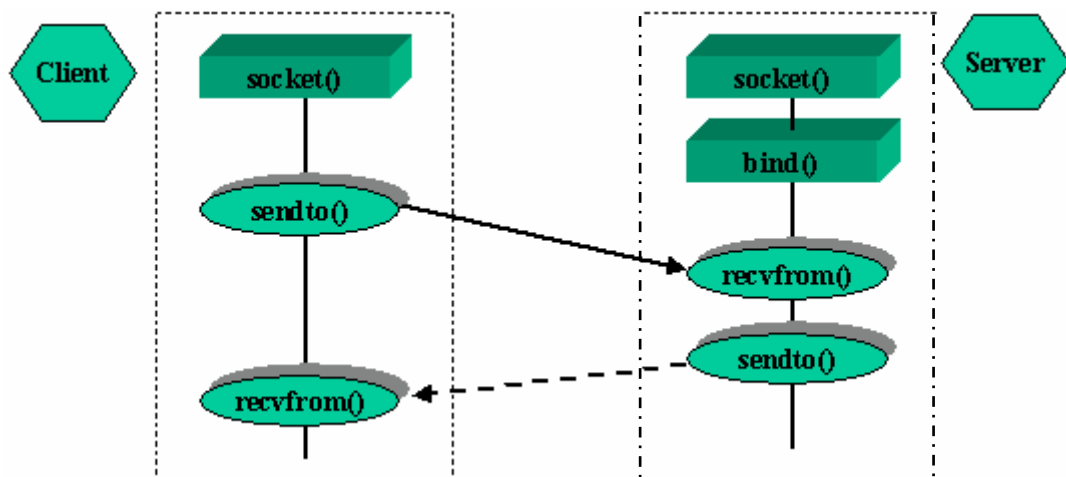


- `socket()`: Server yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng giao vận.
- `bind()`: Server yêu cầu gán số hiệu cổng cho socket..

Giai đoạn 2: Client tạo Socket.



Giai đoạn 3: Trao đổi thông tin giữa Client và Server.



Sau khi tạo Socket xong, Client và Server có thể trao đổi thông tin qua lại với nhau thông qua hai hàm `sendto()` và `recvfrom()`. Đơn vị dữ liệu trao đổi giữa Client và Server là các **Datagram Package** (Gói tin thư tín). Protocol của ứng dụng phải định nghĩa khuôn dạng và ý nghĩa của các Datagram Package. Mỗi Datagram Package có chứa thông tin về địa chỉ người gửi và người nhận (IP, Port).

1.3. Socket dưới ngôn ngữ Java

Java hỗ trợ lập trình mạng thông qua các lớp trong gói **java.net**. Một số lớp tiêu biểu được dùng cho lập trình Client-Server sử dụng socket làm phương tiện giao tiếp như:

- `InetAddress`: Lớp này quản lý địa chỉ Internet bao gồm địa chỉ IP và tên máy tính.
- `Socket`: Hỗ trợ các phương thức liên quan đến Socket cho chương trình Client ở chế độ có kết nối.
- `ServerSocket`: Hỗ trợ các phương thức liên quan đến Socket cho chương trình Server ở chế độ có kết nối.
- `DatagramSocket`: Hỗ trợ các phương thức liên quan đến Socket ở chế độ không kết nối cho cả Client và Server.
- `DatagramPacket`: Lớp cài đặt gói tin dạng thư tín người dùng (Datagram Packet) trong giao tiếp giữa Client và Server ở chế độ không kết nối.

1.3.1. Xây dựng chương trình Client ở chế độ có kết nối

Các bước tổng quát:

- Mở một socket kết nối đến server đã biết địa chỉ IP (hay tên miền) và số hiệu cổng.
- Lấy `InputStream` và `OutputStream` gắn với `Socket`.
- Tham khảo Protocol của dịch vụ để định dạng đúng dữ liệu trao đổi với Server.
- Trao đổi dữ liệu với Server nhờ vào các `InputStream` và `OutputStream`.
- Đóng `Socket` trước khi kết thúc chương trình.

1.3.1.1. Lớp `java.net.Socket`

Lớp `Socket` hỗ trợ các phương thức cần thiết để xây dựng các chương trình client sử dụng socket ở chế độ có kết nối. Dưới đây là một số phương thức thường dùng để xây dựng Client:

`public Socket(String HostName, int PortNumber) throws IOException`

Phương thức này dùng để kết nối đến một server có tên là `HostName`, cổng là `PortNumber`. Nếu kết nối thành công, một kênh ảo sẽ được hình thành giữa Client và Server.

- `HostName`: Địa chỉ IP hoặc tên logic theo dạng tên miền.
- `PortNumber`: có giá trị từ 0 ..65535

Ví dụ: Mở socket và kết nối đến Web Server của khoa Công nghệ Thông tin, Đại học Cần Thơ:

```
Socket s = new Socket("www.cit.ctu.edu.vn",80);
```

```
Hoặc: Socket s = new Socket("203.162.36.149",80);
```

`public InputStream getInputStream()`

Phương thức này trả về `InputStream` nối với `Socket`. Chương trình Client dùng `InputStream` này để nhận dữ liệu từ Server gửi về.

Ví dụ: Lấy `InputStream` của `Socket s`: `InputStream is = s.getInputStream();`

`public OutputStream getOutputStream()`

Phương thức này trả về `OutputStream` nối với `Socket`. Chương trình Client dùng `OutputStream` này để gửi dữ liệu cho Server.

Ví dụ: Lấy `OutputStream` của `Socket s`: `OutputStream os = s.getOutputStream();`

`public close()`

Phương thức này sẽ đóng `Socket` lại, giải phóng kênh ảo, xóa kết nối giữa Client và Server.

Ví dụ: Đóng `Socket s`:

```
s.close();
```


1.3.1.2. Chương trình TCPEchoClient

Trên hệ thống UNIX, Dịch vụ Echo được thiết kế theo kiến trúc Client-Server sử dụng Socket làm phương tiện giao tiếp. Cổng mặc định dành cho Echo Server là 7, bao gồm cả hai chế độ có kết nối và không kết nối.

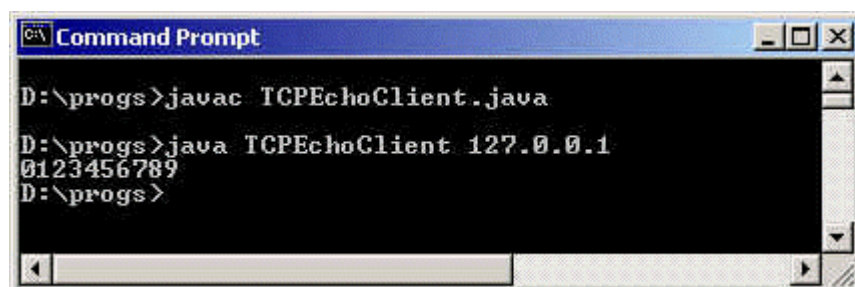
Chương trình TCPEchoClient sẽ kết nối đến EchoServer ở chế độ có kết nối, lần lượt gửi đến Echo Server 10 ký tự từ '0' đến '9', chờ nhận kết quả trả về và hiển thị chúng ra màn hình.

Hãy lưu chương trình sau vào tập tin TCPEchoClient.java

```
import java.io.*;
import java.net.Socket;

public class TCPEchoClient{
    public static void main(String args[]){
        try {
            Socket s = new Socket(args[0],7);          // Kết nối đến Server
            InputStream is = s.getInputStream(); // Lấy InputStream
            OutputStream os = s.getOutputStream(); // Lấy OutputStream
            for (int i='0'; i<='9';i++){              // Gửi '0' -> '9' đến EchoServer
                os.write(i);                          // Gửi 1 ký tự sang Server
                int ch = is.read();                    // Chờ nhận 1 ký tự từ Server
                System.out.print((char)ch);           // In ký tự nhận được ra màn hình
            }
        } catch(IOException ie){
            System.out.println("Lỗi: Không tạo được socket");
        }
    }
}
```

Biên dịch và thực thi chương trình như sau:



```
Command Prompt
D:\progs>javac TCPEchoClient.java
D:\progs>java TCPEchoClient 127.0.0.1
0123456789
D:\progs>
```

Chương trình này nhận một đối số là địa chỉ IP hay tên miền của máy tính mà ở đó Echo Server đang chạy. Trong hệ thống mạng TCP/IP mỗi máy tính được gán một địa chỉ IP cục bộ là **127.0.0.1** hay có tên là **localhost**. Trong ví dụ trên, chương trình Client kết nối đến Echo Server trên cùng máy với nó.

1.3.2. Xây dựng chương trình Server ở chế độ có kết nối

1.3.2.1. Lớp java.net.ServerSocket

Lớp ServerSocket hỗ trợ các phương thức cần thiết để xây dựng các chương trình Server sử dụng socket ở chế độ có kết nối. Dưới đây là một số phương thức thường dùng để xây dựng Server:

public ServerSocket(int PortNumber);

Phương thức này tạo một Socket với số hiệu cổng là PortNumber mà sau đó Server sẽ lắng nghe trên cổng này.

Ví dụ: Tạo socket cho Server với số hiệu cổng là 7:

```
ServerSocket ss = new ServerSocket(7);
```

public Socket accept();

Phương thức này lắng nghe yêu cầu kết nối của các Client. Đây là một phương thức hoạt động ở chế độ nghẽn. Nó sẽ bị nghẽn cho đến khi có một yêu cầu kết nối của client gửi đến.

Khi có yêu cầu kết nối của Client gửi đến, nó sẽ chấp nhận yêu cầu kết nối, trả về một Socket là một đầu của kênh giao tiếp ảo giữa Server và Client yêu cầu kết nối.

Ví dụ: Socket ss chờ nhận yêu cầu kết nối:

```
Socket s = ss.accept();
```

Server sau đó sẽ lấy InputStream và OutputStream của Socket mới s để giao tiếp với Client.

1.3.2.2. Xây dựng chương trình Server phục vụ tuần tự

Một Server có thể được cài đặt để phục vụ các Client theo hai cách: phục vụ tuần tự hoặc phục vụ song song.

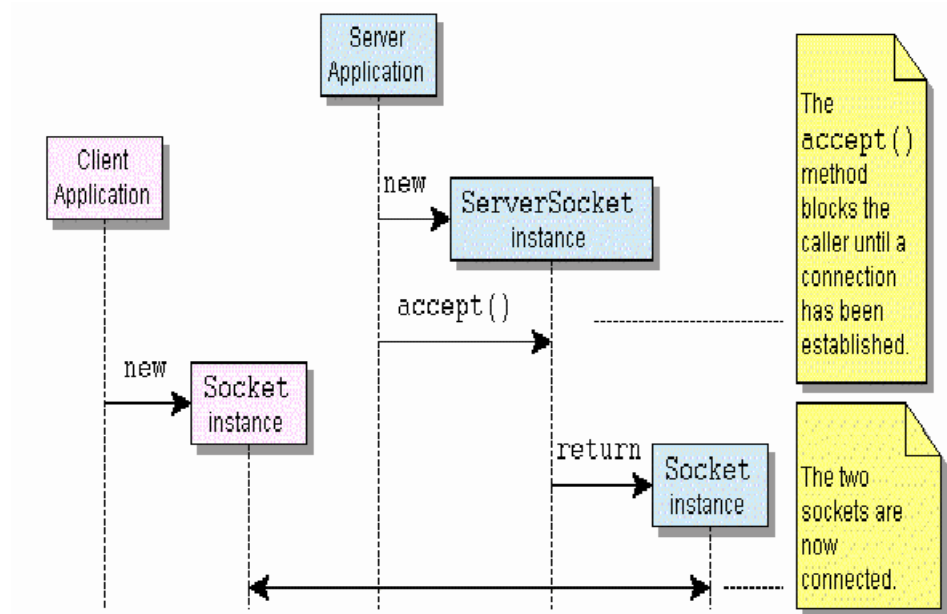
Trong chế độ phục vụ tuần tự, tại một thời điểm Server chỉ chấp nhận một yêu cầu kết nối. Các yêu cầu kết nối của các Client khác đều không được đáp ứng (đưa vào hàng đợi).

Ngược lại trong chế độ phục vụ song song, tại một thời điểm Server chấp nhận nhiều yêu cầu kết nối và phục vụ nhiều Client cùng lúc.

Các bước tổng quát của một Server phục vụ tuần tự

- a. Tạo socket và gán số hiệu cổng cho server.
- b. Lắng nghe yêu cầu kết nối.
- c. Với một yêu cầu kết nối được chấp nhận thực hiện các bước sau:
 - o Lấy InputStream và OutputStream gắn với Socket của kênh ảo vừa được hình thành.
 - o Lặp lại công việc sau:
 - Chờ nhận các yêu cầu (công việc).
 - Phân tích và thực hiện yêu cầu.
 - Tạo thông điệp trả lời.
 - Gửi thông điệp trả lời về Client.

- Nếu không còn yêu cầu hoặc Client kết thúc, đóng Socket và quay lại bước 2.



1.3.2.3. Chương trình STCPEchoServer

STCPEchoServer cài đặt một Echo Server phục vụ tuần tự ở chế độ có kết nối. Server lắng nghe trên cổng mặc định số 7.

Hãy lưu chương trình sau vào tập tin STCPEchoServer.java

```

import java.net.*;
import java.io.*;

public class STCPEchoServer {
    public final static int defaultPort = 7;
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(defaultPort);
            while (true) {
                try {
                    Socket s = ss.accept();
                    OutputStream os = s.getOutputStream();
                    InputStream is = s.getInputStream();
                    int ch=0;
                    while(true) {
                        ch = is.read();
                        if(ch == -1) break;
                        os.write(ch);
                    }
                    s.close();
                }
            }
        }
    }
}
  
```

```

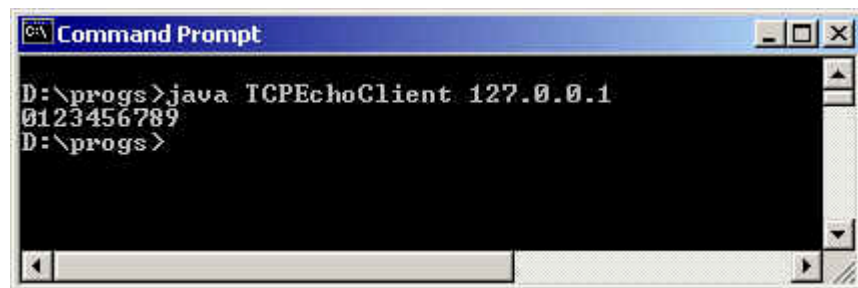
        } catch (IOException e) {
            System.err.println(" Connection Error: "+e);
        }
    }
} catch (IOException e) {
    System.err.println(" Server Creation Error:"+e);
}
}
}

```

Biên dịch và thực thi chương trình theo cách sau:



Mở một cửa sổ DOS khác và thực thi chương trình TCPEchoClient ta có kết quả như sau:



Hai chương trình này có thể nằm trên hai máy khác nhau. Trong trường hợp đó khi thực hiện chương trình TCPEchoClient phải chú ý nhập đúng địa chỉ IP của máy tính đang chạy chương trình STCP EchoServer.

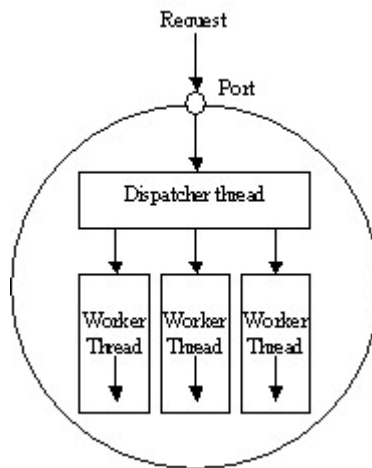
Xem địa chỉ IP của một máy tính Windows bằng lệnh **ipconfig**.

1.3.2.4. Server phục vụ song song

Các bước tổng quát của một Server phục vụ song song

Server phục vụ song song gồm 2 phần thực hiện song song nhau:

- Phần 1: Xử lý các yêu cầu kết nối.
- Phần 2: Xử lý các thông điệp yêu cầu từ khách hàng.



Hình 4.5: Server ở chế độ song song

Trong đó **Phần 1** là (Dispatcher Thread), **Phần 2** là các (Worker Thread)

Phần 1: Lặp lại các công việc sau:

- Lắng nghe yêu cầu kết nối của khách hàng.
- Chấp nhận một yêu cầu kết nối.
 - Tạo kênh giao tiếp ảo mới với khách hàng.
 - Tạo Phần 2 để xử lý các thông điệp yêu cầu của khách hàng.

Phần 2: Lặp lại các công việc sau:

- Chờ nhận thông điệp yêu cầu của khách hàng.
- Phân tích và xử lý yêu cầu.
- Gởi thông điệp trả lời cho khách hàng.

Phần 2 sẽ kết thúc khi kênh ảo bị xóa đi.

Với mỗi Client, trên Server sẽ có một **Phần 2** để xử lý yêu cầu của khách hàng. Như vậy tại một thời điểm bất kỳ luôn tồn tại 1 **Phần 1** và 0 hoặc nhiều **Phần 2**.

Do Phần 2 thực thi song song với Phần 1 cho nên nó được thiết kế là một Thread.

1.3.2.5. Chương trình PTCPEchoServer

PTCPEchoServer cài đặt một Echo Server phục vụ song song ở chế độ có kết nối. Server lắng nghe trên cổng mặc định là 7. Chương trình này gồm 2 lớp:

- Lớp TCPEchoServer, cài đặt các chức năng của Phần 1 - xử lý các yêu cầu kết nối của TCPEchoClient.
- Lớp RequestProcessing, là một Thread cài đặt các chức năng của Phần 2 - Xử lý các thông điệp yêu cầu.

Hãy lưu chương trình sau vào tập tin PTCPEchoServer.java

```

import java.net.*;
import java.io.*;

public class PTCPEchoServer {

    public final static int defaultPort = 7; // Cổng mặc định
    public static void main(String[] args) {

```

```

try {
    //Tạo socket cho server
    ServerSocket ss = new ServerSocket(defaultPort);
    while (true) {
        try {
            Socket s = ss.accept(); // Lắng nghe các yêu cầu kết nối
            // Tạo phân xử lý
            RequestProcessing rp = new RequestProcessing(s);
            rp.start(); // Khởi động phân xử lý cho Client hiện tại
        } catch (IOException e) {
            System.out.println("Connection Error: "+e);
        }
    }
} catch (IOException e) {
    System.err.println("Create Socket Error: "+e);
}
}

class RequestProcessing extends Thread {
    Socket channel; //Socket của kênh ảo nối với Client hiện tại
    public RequestProcessing(Socket s){
        channel = s; // Nhận socket của kênh ảo nối với Client
    }
    public void run() {
        try {
            OutputStream os = channel.getOutputStream();
            InputStream is = channel.getInputStream();
            while (true) {
                int n = is.read();    // Nhận ký tự từ Client
                if (n == -1) break;    // Thoát nếu kênh ảo bị xóa
                os.write(n);    // Gởi ký tự nhận được về Client
            }
        } catch (IOException e) {
            System.err.println("Request Processing Error: "+e);
        }
    }
}

```

}

Biên dịch và thực thi chương trình như sau:



Sau đó mở thêm 2 cửa sổ DOS khác để thực thi chương trình TCPEchoClient kết nối tới PTCEchoServer. Ta sẽ nhận thấy rằng PTCEchoServer có khả năng phục vụ đồng thời nhiều Client.

1.3.3. Xây dựng chương trình Client - Server ở chế độ không kết nối

Khi sử dụng socket, ta có thể chọn giao thức UDP cho lớp giao vận. UDP viết tắt của User Datagram Protocol, cung cấp cơ chế giao vận không bảo đảm và không kết nối trên mạng IP, ngược với giao thức giao vận tin cậy, có kết nối TCP.

Cả giao thức TCP và UDP đều phân dữ liệu ra thành các gói tin. Tuy nhiên TCP có thêm vào những tiêu đề (Header) vào trong gói tin để cho phép truyền lại những gói tin thất lạc và tập hợp các gói tin lại theo thứ tự đúng đắn. UDP không cung cấp tính năng này, nếu một gói tin bị thất lạc hoặc bị lỗi, nó sẽ không được truyền lại, và thứ tự đến đích của các gói tin cũng không giống như thứ tự lúc nó được gửi đi.

Tuy nhiên, về tốc độ, UDP sẽ truyền nhanh gấp 3 lần TCP. Cho nên chúng thường được dùng trong các ứng dụng đòi hỏi thời gian truyền tải ngắn và không cần tính chính xác cao, ví dụ truyền âm thanh, hình ảnh . . .

Mô hình client - server sử dụng lớp ServerSocket và Socket ở trên sử dụng giao thức TCP. Nếu muốn sử dụng mô hình client - server với giao thức UDP, ta sử dụng hai lớp `java.net.DatagramSocket` và `java.net.DatagramPacket`.

`DatagramSocket` được sử dụng để truyền và nhận các `DatagramPacket`. Dữ liệu được truyền đi là một mảng những byte, chúng được gói vào trong lớp `DatagramPacket`. Chiều dài của dữ liệu tối đa có thể đưa vào `DatagramPacket` là khoảng 60.000 byte (phụ thuộc vào dạng đường truyền). Ngoài ra `DatagramPacket` còn chứa địa chỉ IP và cổng của quá trình gửi và nhận dữ liệu.

Cổng trong giao thức TCP và UDP có thể trùng nhau. Trên cùng một máy tính, bạn có thể gán cổng 20 cho socket dùng giao thức TCP và cổng 20 cho socket sử dụng giao thức UDP.

1.3.3.1. Lớp `DatagramPacket`

Lớp này dùng để đóng gói dữ liệu gửi đi. Dưới đây là các phương thức thường sử dụng để thao tác trên dữ liệu truyền / nhận qua `DatagramSocket`.

`public DatagramPacket(byte[] b, int n)`

- Là phương thức khởi tạo, cho phép tạo ra một `DatagramPacket` chứa **n** bytes dữ liệu đầu tiên của mảng **b**. (n phải nhỏ hơn chiều dài của mảng b)

- Phương thức trả về một đối tượng thuộc lớp DatagramPacket

Ví dụ: Tạo DatagramPacket để nhận dữ liệu:

```
Byte buff[] = new byte[60000]; // Nơi chứa dữ liệu nhận được
```

```
DatagramPacket inPacket = new DatagramPacket(buff, buff.length);
```

public DatagramPacket(byte[] b, int n, InetAddress ia, int port)

- Phương thức này cho phép tạo một DatagramPacket chứa dữ liệu và cả địa chỉ của máy nhận dữ liệu.
- Phương thức trả về một đối tượng thuộc lớp DatagramPacket

Ví dụ: Tạo DatagramPacket chứa chuỗi "My second UDP Packet", với địa chỉ máy nhận là www.cit.ctu.edu.vn, cổng của quá trình nhận là 19:

```
try { //Địa chỉ Internet của máy nhận
    InetAddress ia = InetAddress.getByName("www.cit.ctu.edu.vn");
    int port = 19; // Cổng của socket nhận
    String s = "My second UDP Packet"; // Dữ liệu gửi đi
    byte[] b = s.getBytes(); // Đổi chuỗi thành mảng bytes
    // Tạo gói tin gửi đi
    DatagramPacket outPacket = new DatagramPacket(b, b.length, ia, port);
} catch (UnknownHostException e) {
    System.err.println(e);
}
```

Các phương thức lấy thông tin trên một DatagramPacket nhận được

Khi nhận được một DatagramPacket từ một quá trình khác gửi đến, ta có thể lấy thông tin trên DatagramPacket này bằng các phương thức sau:

- public synchronized() InetAddress getAddress(): Địa chỉ máy gửi
- public synchronized() int getPort(): Cổng của quá trình gửi
- public synchronized() byte[] getData(): Dữ liệu từ gói tin
- public synchronized() int getLength(): Chiều dài của dữ liệu trong gói tin

Các phương thức đặt thông tin cho gói tin gửi

Trước khi gửi một DatagramPacket đi, ta có thể đặt thông tin trên DatagramPacket này bằng các phương thức sau:

- public synchronized() void setAddress(InetAddress dis): Đặt địa chỉ máy nhận.
- public synchronized() void setPort(int port): Đặt cổng quá trình nhận
- public synchronized() void setData(byte buffer[]): Đặt dữ liệu gửi
- public synchronized() void setLength(int len): Đặt chiều dài dữ liệu gửi

1.3.3.2. Lớp DatagramSocket

Lớp này hỗ trợ các phương thức sau để gửi / nhận các DatagramPacket

public DatagramSocket() throws SocketException

Tạo Socket kiểu không kết nối cho Client. Hệ thống tự động gán số hiệu cổng chưa sử dụng cho socket.

Ví dụ: Tạo một socket không kết nối cho Client:

```
try{  
    DatagramSocket ds = new DatagramSocket();  
} catch (SocketException se) {  
    System.out.print("Create DatagramSocket Error: "+se);  
}
```

public DatagramSocket(int port) throws SocketException

Tạo Socket kiểu không kết nối cho Server với số hiệu cổng được xác định trong tham số (port).

Ví dụ: Tạo một socket không kết nối cho Server với số hiệu cổng là 7:

```
try{  
    DatagramSocket dp = new DatagramSocket(7);  
} catch (SocketException se) {  
    System.out.print("Create DatagramSocket Error: "+se);  
}
```

public void send(DatagramPacket dp) throws IOException

Dùng để gửi một DatagramPacket đi.

Ví dụ: Gửi chuỗi "My second UDP Packet", cho quá trình ở địa chỉ www.cit.ctu.edu.vn, cổng nhận là 19:

```
try {  
    DatagramSocket ds = new DatagramSocket(); //Tạo Socket  
    //Địa chỉ Internet của máy nhận  
    InetAddress ia = InetAddress.getByName("www.cit.ctu.edu.vn");  
    int port = 19; // Cổng của quá trình nhận  
    String s = "My second UDP Packet"; // Dữ liệu cần gửi  
    byte[] b = s.getBytes(); // Đổi sang mảng bytes  
    // Tạo gói tin  
    DatagramPacket outPacket = new DatagramPacket(b, b.length, ia, port);  
    ds.send(outPacket); // Gửi gói tin đi  
} catch (IOException e) {  
    System.err.println(e);  
}
```

Public synchronized void receive(DatagramPacket dp) throws IOException

Chờ nhận một DatagramPacket. Quá trình sẽ bị nghẽn cho đến khi có dữ liệu đến.

Ví dụ:

```
try {
    DatagramSocket ds = new DatagramSocket(); //Tạo Socket
    byte[] b = new byte[60000];              // Nơi chứa dữ liệu nhận được
    DatagramPacket inPacket = new DatagramPacket(b, b.length); // Tạo gói tin
    ds.receive(inPacket);                    // Chờ nhận gói tin
} catch (IOException e) {
    System.err.println(e);
}
```

1.3.3.3. Chương trình UDPEchoServer

Chương trình UDPEchoServer cài đặt Echo Server ở chế độ không kết nối, cổng mặc định là 7. Chương trình chờ nhận từng gói tin, lấy dữ liệu ra khỏi gói tin nhận được và gửi ngược dữ liệu đó về Client.

Lưu chương trình sau vào tập tin UDPEchoServer.java

```
import java.net.*;
import java.io.*;

public class UDPEchoServer {
    public final static int port = 7;    // Cổng mặc định của Server
    public static void main(String[] args) {
        try {
            DatagramSocket ds = new DatagramSocket(port); // Tạo Socket với cổng là 7
            byte[] buffer = new byte[6000]; // Vùng đệm chứa dữ liệu cho gói tin nhận
            while(true) {    // Tạo gói tin nhận
                DatagramPacket incoming = new DatagramPacket(buffer,buffer.length);
                ds.receive(incoming); // Chờ nhận gói tin gửi đến
                // Lấy dữ liệu khỏi gói tin nhận
                String theString = new String(incoming.getData(),0,incoming.getLength());
                // Tạo gói tin gửi chứa dữ liệu vừa nhận được
                DatagramPacket outgoing = new DatagramPacket(theString.getBytes(),
                    incoming.getLength(),incoming.getAddress(), incoming.getPort());
                ds.send(outgoing);
            }
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Biên dịch và thực thi chương trình như sau



1.3.3.4. Chương trình UDPEchoClient

Chương trình này cho phép người sử dụng nhập các chuỗi từ bàn phím, gửi chuỗi sang EchoServer ở chế độ không kết nối ở cổng số 7, chờ nhận và in dữ liệu từ Server gửi về ra màn hình.

Lưu chương trình sau vào tập tin UDPEchoClient.java

```
import java.net.*;
import java.io.*;

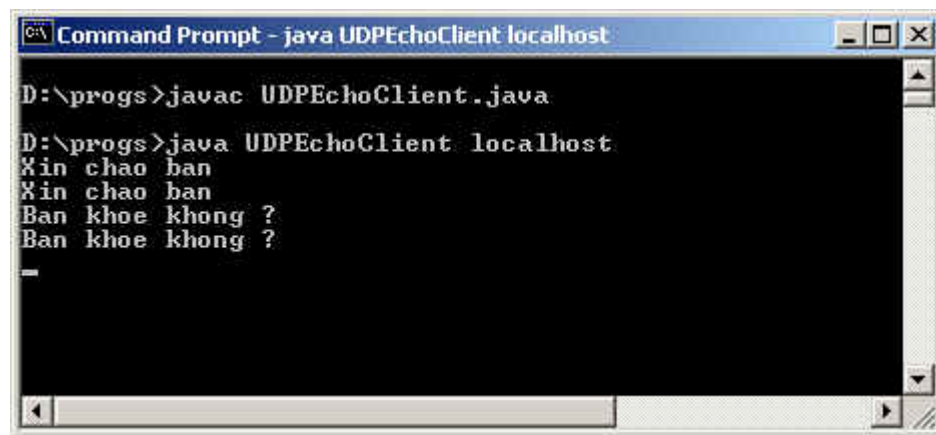
public class UDPEchoClient extends Object{
    public final static int serverPort = 7;    // Cổng mặc định của Echo Server
    public static void main(String[] args) {
        try {
            if (args.length == 0) {    // Kiểm tra tham số, là địa chỉ của Server
                System.out.print("Syntax: java UDPEchoClient HostName");
                return;
            }
            DatagramSocket ds = new DatagramSocket(); // Tạo DatagramSocket
            InetAddress server = InetAddress.getByName(args[0]); // Địa chỉ Server
            while(true) {
                InputStreamReader isr = new InputStreamReader(System.in); // Nhập
                BufferedReader br = new BufferedReader(isr);                // một chuỗi
                String theString = br.readLine();                            // từ bàn phím
                byte[] data = theString.getBytes();                        // Đổi chuỗi ra mảng bytes
                // Tạo gói tin gửi
                DatagramPacket dp = new DatagramPacket(data,data.length,server, serverPort);
                ds.send(dp); // Send gói tin sang Echo Server
                byte[] buffer = new byte[6000];    // Vùng đệm cho dữ liệu nhận
                // Gói tin nhận
                DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
                ds.receive(incoming);    // Chờ nhận dữ liệu từ EchoServer gửi về
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        // Đổi dữ liệu nhận được dạng mảng bytes ra chuỗi và in ra màn hình
        System.out.println(new String(incoming.getData(), 0, incoming.getLength()));
    }
} catch (IOException e) {
    System.err.println(e);
}
}
}
}

```

Biên dịch và thực thi chương trình như sau:



```

C:\ Command Prompt - java UDPEchoClient localhost
D:\progs>javac UDPEchoClient.java
D:\progs>java UDPEchoClient localhost
Xin chao ban
Xin chao ban
Ban khoe khong ?
Ban khoe khong ?
_

```

Chú ý, khi thực hiện chương trình UDPEchoClient phải đưa vào đối số là địa chỉ của máy tính đang thực thi chương trình UDPEchoServer. Trong ví dụ trên, Server và Client cùng chạy trên một máy nên địa chỉ của UDPEchoServer là **localhost** (hay 127.0.0.1). Nếu UDPEchoServer chạy trên máy tính khác thì khi thực thi, ta phải biết được địa chỉ IP của máy tính đó và cung cấp vào đối số của chương trình. Chẳng hạn, khi UDPEchoServer đang phục vụ trên máy tính ở địa chỉ 172.18.250.211, ta sẽ thực thi UDPEchoClient theo cú pháp sau:

```
java UDPEchoClient 172.18.250.211
```