TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI



BÁO CÁO BÀI TẬP LỚN

MẠNG MÁY TÍNH

ĐỀ TÀI: Tìm hiểu mô hình lập trình mạng Socket trong Java. Sử dụng để viết chương trình truyền file đơn giữa 2 máy. Trình bày phân tích thiết kế.

Giáo viên hướng dẫn: NGUYỄN ĐỨC TOÀN

Sinh viên:KHIẾU VĂN NGUYÊN MSSV:20168756

ĐỒNG VĂN THIỆN MSSV:20168509

NGUYỄN THỊ THUỶ MSSV:20168524

MỤC LỤC

Lời nói đầu…………………………………………………3

1.Khái niệm socket……………………………………………….3

1.1.Lịch sử hình thành…………………………………………...3

1.2.Nguyên lý hoạt động………………………………………..10

2.Socket trong JAVA…………………………………………….13

2.1.Xác định chương trình client ở chế độ có nối kết………………………………………………………………….13

2.2.Xác định chương trình server ở chế độ có nối kết………………………………………………………………….14

3.Phân tích thiết kế……………………………………………...18

KẾT LUẬN…………………………………………………………20

**LỜI NÓI ĐẦ**U

Hiện nay vấn đề toàn cầu hóa thông tin và tốc độ phát triển của khoa học công nghệ diễn ra một cách nhanh chóng, một kỉ nguyên mới mới được mở ra, kỉ nguyên của xã hội hóa thông tin. Công nghệ thông tin và truyền thông phát triển đã đưa thế giới chuyển sang thời đại mới thời đại của công nghệ thông tin. Việc năm bắt và ứng dụng Công nghệ thông tin trong các lĩnh vực khoa học, kinh tế, xã hội, đã đem lại cho các danh nghiệp và các tổ chức những thành tựu và lợi ích to lớn.

Máy tính đã trở thành công cụ đắc lực và không thể thiếu của con người. Các tổ chức, công ty hay các cơ quan cần phải xây dựng hệ thống máy tính cho riêng mình để trao đổi dữ liệu giữa các bộ phận. Dữ liệu được truyền đi trên mạng phải đảm bảo: dữ liệu được truyền tới đích nhanh chóng và đúng đắn. Hầu hết dữ liệu được truyền qua mạng là dưới dạng truyền file.

Nhằm tìm hiểu thấu đáo một trong số các phướng pháp truyền file qua mạng nhóm em chọn đề tài “ Tìm hiểu mô hình lập trình mạng Socket trong java. Sử dụng để viết chương trình truyền file đơn giản giữa hai máy”.

**Lập trình Socket trong java**

**1.Khái niệm Socket**

**1.1.Lịch sử hình thành**

- Khái niệm Socket xuất hiện lần đầu tiên vào khoảng năm 1980 tại trường đại học Berkeley Mỹ. Đó là một chương trình được thiết kế để giúp máy tính nối mạng ở khắp mọi nơi có thể trao đổi thông tin với nhau. Lúc đầu có được sử dụng trên các máy Unix và có tên gọi là Berkeley Socket Interface.

- Tiếp đó cùng với sự phát triển của các ứng dụng mạng, socket được hỗ trợ trong nhiều ngôn ngữ lập trình và chạy trên nhiều nền tảng hệ điều hành khác nhau. Ví dụ như WinSock dùng cho các ứng dụng của Microsoft, Socket++ dùng cho các lập trình viên sử dụng Unix…

- Có câu hỏi đặt ra là tại sao chúng ta lại sử dụng Socket trong truyền thống giữa các máy tính. Để trả lời câu hỏi này chúng ta phải quay lại thời điểm trước khi Socket ra đời:

Trong thời kì này trên hệ thống Unix việc vào/ra dữ liệu được thực hiện theo mô hình 3 bước Open-Read/Write-Close. Để thực hiện việc vào ra dữ liệu trước hết chương trình phải tạo ra một kết nối với tài nguyên mà nó muốn giao tiếp(tài nguyên này có thể là bàn phím, bộ nhớ trong, file…), sau khi kết nối đã được thực hiện, chương trình có thể trao đổi dữ liệu thông qua các thao tác Read- đưa dữ liệu từ tài nguyên đã kết nối vào chương trình để xử lý hoặc Write- đưa dữ liệu đã xử lý từ chương trình ra tài nguyên. Một ví dụ điển hình cho kiểu vào/ra này là thao tác với file dữ liệu mà chúng ta khá quen thuộc trong các ngôn ngữ lập trình: Khi người lập trình muốn thao tác với một file dữ liệu họ tiến hành như sau:

+ Mở file cần sử dụng với các quyền thích hợp trên đó

+ Thực hiện việc đọc dữ liệu từ file để xử lý hay đưa dữ liệu để xử lý để ghi vào file.

+ Đóng file sau khi đã sử dụng xong.

- Khi việc trao đổi dữ liệu giữa các chương trình và kết nối mạng được đưa vào hệ thống Unix người ta mong muốn việc trao đổi dữ liệu giữa các chương trình cũng sẽ được thực hiện theo mô hình ba bước của vào/ra dữ liệu nhằm tránh cho người lập trình những khó khăn khi giao tiếp với các tầng bên dưới tầng ứng dụng. Để làm được điều đó, socket được sử dụng. Khi hai chương trình muốn giao tiếp với nhau, mỗi chương trình sẽ tạo ra một socket, chúng đóng vai trò là các điểm cuối trong một kết nối và thực hiện trao đổi thông tin giữa hai chương trình. Đối với người lập trình, socket được xem như một tài nguyên hệ thống mà chương trình cần giao tiếp nên chương trình có thể thực hiện giao tiếp với socket theo mô hình ba bước giống như việc vào/ra dữ liệu. Như vậy sự ra đời của socket gắn liền với nhu cầu truyền thông máy tính. Sau đây chúng ta sẽ đưa ra định nghĩa cụ thể về socket.

***Định nghĩa :***

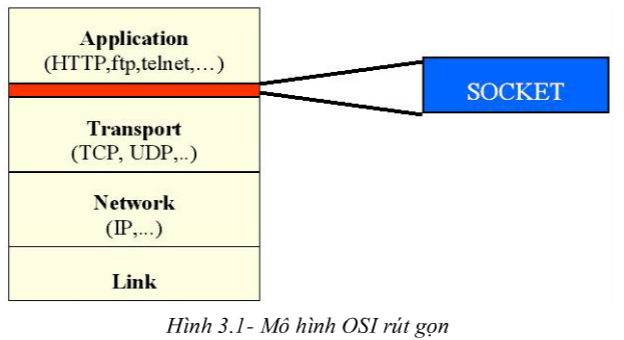
- Có nhiều định nghĩa khác nhau về socket tùy theo cách nhìn của người sử dụng.

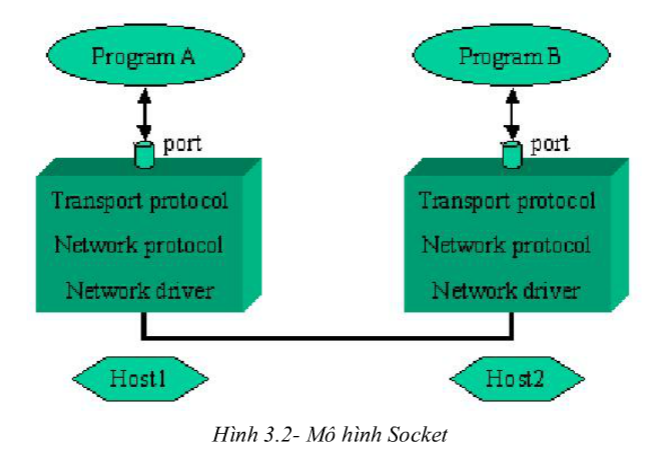
- Một cách tổng quát nhất có thể định nghĩa: *Một Socket là một điểm cuối trong một kết nối giữa hai chương trình đang chạy trên mạng*

- Nhìn trên quan điểm của người phát triển ứng dụng người ta có thể định nghĩa *Socket là một phương pháp để thiết lập kết nối truyền thông giữa một chương trình yêu cầu dịch vụ (được gán nhãn là Client) và một chương trình cung cấp dịch vụ (được gán nhãn là server) trên mạng hoặc trên cùng một máy tính.*

- Đối với người lập trình, họ nhìn nhận Socket như một giao diện nằm giữa tầng ứng dụng và tầng khác trong mô hình mạng OSI có nhiệm vụ thực hiện việc giao tiếp giữa chương trình ứng dụng với các tầng bên dưới của mạng

.

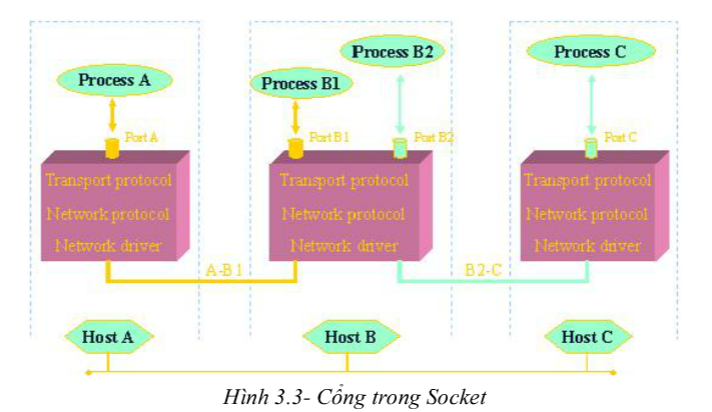




- Tuy nhiên, các lập trình viên hiện nay gần như luôn luôn bị ngăn cản tạo socket riêng bằng cách thủ công bởi dù bạn dùng Java, serlet hay CGI, PHP… có thể bạn sẽ không bao giờ mở được cổng một cách tường minh. Thay vào đó các lập trình viên sử dụng thư viện socket được hỗ trợ sẵn bởi các ngôn ngữ lập trình. Như vậy các socket vẫn tồn tại để kết nối các ứng dụng của người dùng, nhưng các chi tiết của socket được ẩn trong những lớp sâu hơn để mọi người không phải động chạm đến.

***\*Số hiệu cổng của Socket:***

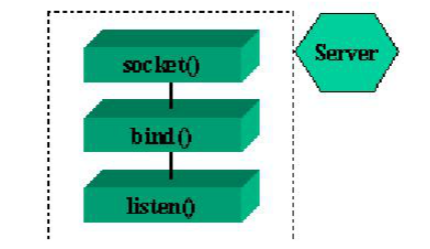
- Để có thể thực hiện các cuộc giao tiếp, một trong hai quá trình phải công bố số hiệu cổng của socket mà mình sử dụng. Mỗi cổng giao tiếp thể hiện một địa chỉ xác định trong hệ thống. Khi quá trình được gán một số hiệu cổng, nó có thể nhận dữ liệu gửi đến cổng này từ các quá trình khác. Quá trình còn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



Trong hình trên, địa chỉ của quá trình B1 được xác định bằng hai thông tin (Host B, Port B1): Địa chỉ máy tính có thể là địa chỉ IP dạng 203.162.88.162 hay là địa chỉ cho dạng trên miền như www.hpu.edu.vn

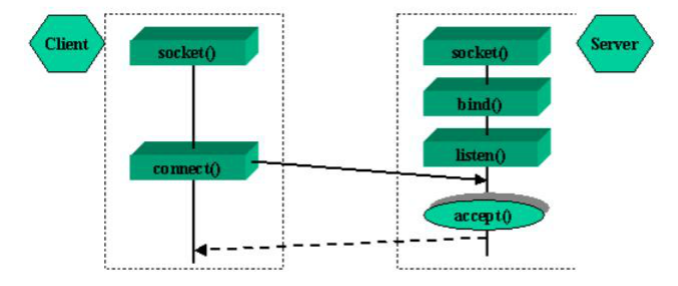
Số hiệu cổng gán cho Socket phải duy nhất trên phạm vi máy tính đó, có giá trị trong khoảng từ 0 đến 65535 (16 bit). Trong thực tế thì các số hiệu cổng từ 0 đến 1023 (gồm có 1024 cổng) đã dành cho các dịch vụ nổi tiếng như: http: 80, telnet:21, ftp:23,…. Nếu chúng ta không phải là người quản trị thì nên dùng từ cổng 1024 trở lên.

*- Giai đoạn 1:* Server tạo Socket, gán số hiệu vào 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 vận chuyển.

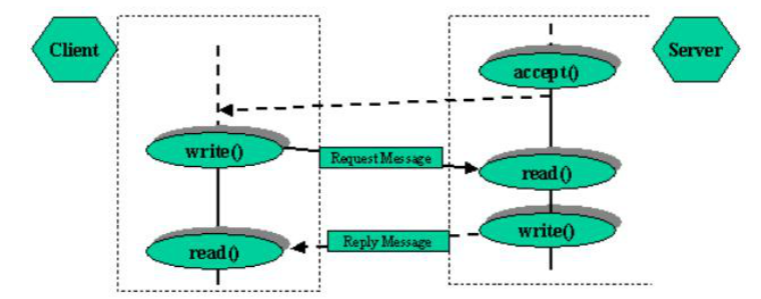
*Bind()*: Server yêu cầu gán số hiệu cổng (port) cho socket. *Listen()*: Server lắng nghe các yêu cầu nối kết từ các client trên cổng đã được gán.

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

*Socket()* : Client yêu cầu tạo một socket để có thể sử dụng các dịch vụ của tầng vận chuyển, thông thường hệ thống tự động gán một số hiệu cổng chưa sử dụng cho socket của Client. *Connect()* : Client gửi yêu cầu nối kết đến server có địa chỉ IP và Port xác định.

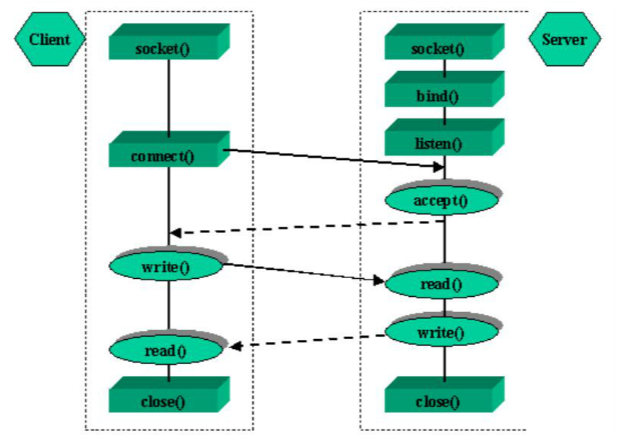
*Accept()* : Server chấp nhận nối kết của client, khi đó một kênh giao tiếp ảo được hình thành, client và server có thể trao đổi thông tin với nhau thông qua kênh ảo này .

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



Sau khi chấp nhận yêu cầu nối kết, thông thường server thực hiện lệnh *read()* và nghẽ cho đến khi có thông điệp yêu cầu(Reaquest 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 (Reply Message) 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 các thông điệp, quy 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 quy định của giao thức. Ngược lại, nếu chúng ta phát triển một ứng dụng clients/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



**1.2.Nguyên lý hoạt động**

- Ta đã thấy khi hai ứng dụng muốn trao đổi dữ liệu qua mạng, chúng sẽ tạo ra ở mỗi phía một socket và trao đổi dữ liệu bằng cách đọc/ghi từ socket. Để hiểu rõ cách thức socket trao đổi dữ liệu chúng ta hãy xem xét nguyên lý hoạt động của chúng.

- Trước hết chúng ta hãy xem xét làm thế nào các socket có thể xác định được nhau. Khi một chương trình tạo ra một socket, một định danh dạng số (định danh dạng số này còn được gọi là số hiệu cổng) sẽ được gán cho socket. Việc gán số hiệu cổng này cho socket có thể được thực hiện bởi chương trình hoặc hệ điều hành tùy theo cách socket được sử dụng như thế nào. Trong mỗi gói tin mà socket gửi đi có chứa hai thông tin để xác định đích đến của gói tin:

+ Một địa chỉ mạng để xác định hệ thống sẽ nhận gói tin.

+ Một số định danh cổng để nói cho hệ thống đích biết socket nào trên nó sẽ nhận dữ liệu.

- Nhờ hai thông tin này mà gói tin có thể đến được đúng máy tính chứa socket mà nó cần đến (nhờ địa chỉ mạng) và được phân phối đến đúng socket đích (nhờ địa chỉ cổng của socket đích).

- Dưới góc độ lập trình các socket thường làm việc theo cặp, một socket đóng vai trò làm server còn các socket khác đóng vai trò như clients. Socket phía server xác định một cổng cho giao tiếp mạng, sau đó chờ nghe yêu cầu mà client gửi tới nó bằng client socket. Do đó các cổng cho server socket phải được biết bởi các chương trình client. Ví dụ server FTP sử dụng một socket để nghe tại cổng 21 do đó nếu một chương trình client muốn giao tiếp với server FTP nó cần phải kết nối đến socket có số hiệu cổng 21.

- Như vậy số hiệu cổng của socket phía server được xác định bởi chương trình, ngược lại cổng cho client socket được xác định bởi hệ điều hành. Khi một socket phía client gửi một gói tin tới socket phía server thì trong gói tin đã có chứa thông tin về địa chỉ của hệ thống client và cổng của socket phía client nên server hoàn toàn có thể gửi thông tin phản hồi cho client.

- Chúng ta có thể khái quát quá trình trao đổi dữ liệu thông qua các socket như sau:

+ Chương trình phía server tạo ra một socket, socket này được chương trình gắn với một cổng trên server. Sau khi được tạo ra socket này (ta gọi là socket phía server) sẽ chờ nghe yêu cầu từ phía clients.

+ Khi chương trình phía clients cần kết nối với một server, nó cũng tạo ra một socket, socket này cũng được hệ điều hành gắn với một cổng. Chương trình client sẽ cung cấp cho socket của nó (ta gọi là socket phía client) địa chỉ mạng và cổng của socket phía server và yêu cầu thực hiện kết nối (nếu chương trình định sử dụng giao thức hướng kết nối) hoặc truyền dữ liệu (nếu chương trình sử dụng giao thức không hướng kết nối)

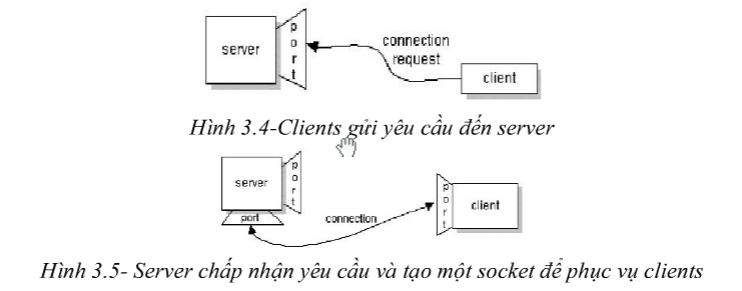
+ Chương trình phía server và chương trình phía clients trao đổi dữ liệu với nhau bằng cách đọc từ socket hoặc ghi vào socket của mình. Các socket ở hai phía nhận dữ liệu từ ứng dụng và đóng gói để gửi đi hoặc nhận các dữ liệu được gửi đến và chuyển cho chương trình ứng dụng bởi socket ở cả hai phía đều biết được địa chỉ mạng và địa chỉ cổng của nhau.

- Ở bước thứ hai chúng ta thấy chương trình ứng dụng phải lựa chọn giao thức mà nó định sử dụng để trao đổi dữ liệu. Tuỳ theo việc chúng ta sử dụng giao thức nào (TCP hay UDP) mà cách thức xử lý trước yêu cầu của clients có thể khác.

- Sau đây chúng ta sẽ xem xét chi tiết cách thức trao đổi dữ liệu của socket với từng loại giao thức.

***\*Socket hỗ trợ TCP* :**

**a**. Ở phía Server: Khi một ứng dụng trên server hoạt động nó sẽ tạo ra một socket và đăng ký với server một cổng ứng dụng và chờ đợi yêu cầu kết nối từ phía clients qua cổng này.



**b**. Ở phía clients: Nó biết địa chỉ của máy trên đó server đang chạy vào cổng và server đang chờ nghe yêu cầu. Do đó khi muốn kết nối đến server, nó cũng tạo một socket chứa địa chỉ máy client và cổng của ứng dụng trên máy clients đồng thời clients sẽ cung cấp cho socket của nó địa chỉ và cổng của server mà nó cần kết nối và yêu cầu socket thực hiện kết nối.

Khi server nhận được yêu cầu kết nối từ clients, nếu nó chấp nhận thì server sẽ sinh ra một socket mới được gắn với một cổng khác với cổng mà nó đang nghe yêu cầu. Sở dĩ server làm như vậy bởi nó cần cổng cũ để tiếp tục nghe yêu cầu từ phía clients trong khi vẫn cần một kết nối với clients.

Sau đó chương trình ứng dụng phía server sẽ gửi thông báo chấp nhận kết nối cho clients cùng thông tin về địa chỉ cổng mới của socket mà nó dành cho clients.

**c**. Quay lại phía clients, nếu kết nối được chấp nhận nghĩa là socket của nó đã được tạo ra thành công và nó có thể sử dụng socket để giao tiếp với server bằng cách viết và ghi tới socket theo cách giao tiếp với một tài nguyên trên máy tính thông thường.

***\*Socket hỗ trợ UDP* :**

**a**. Ở phía Server: Khi một ứng dụng trên server hoạt động nó sẽ tạo ra một socket và đăng ký với server một cổng ứng dụng và chờ đợi yêu cầu kết nối từ phía clients qua cổng này.

**b**. Ở phía Clients: Nó biết địa chỉ của máy trên đó server đang chạy vào cổng và server đang chờ nghe yêu cầu. Do đó khi muốn giao tiếp với server, nó cũng tạo ra một socket chứa địa chỉ máy clients và cổng của ứng dụng trên máy clients đồng thời clients sẽ cung cấp cho socket của nó địa chỉ và cổng của server mà nó cần kết nối. Khi clients muốn gửi tin đế server nó sẽ chuyển dữ liệu cho socket của mình, socket này sẽ chuyển thẳng gói tin mà client muốn gửi tới server dưới dạng một datagram có chứa địa chỉ máy server và cổng mà server đang chờ nghe yêu cầu. Như vậy không hề có một kết nối nào được thực hiện giữa client với server và server cũng không cần tạo ra một socket khác để kết nối với clients thay vào đó server dùng ngay cổng ban đầu để trao đổi dữ liệu.

**2**. **Socket trong java**

Lớp **DatagramPacket**: Lớp cài đặt gói tin dạng Thông qua các lớp trong gói **java.net**, các chương trình Java có thể sử dụng TCP hoặc UDP để giao tiếp qua Internet.

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

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

Một số phương thức cần thiết để xây dựng các chương trình client sử dụng socket ở chế độ có nối kết:

* public **Socket(String HostName, int PortNumber)** : Phương thức này dùng để nối kết đến một server có tên là HostName, cổng là PortNumber. Nếu nối kết 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 giả trị từ 0 ..65535
* public InputStream **getInputStream()** : Phương thức này trả về InputStream nối với Socket. Chương trình Client dùng InputStream này để nhận dữ liệu từ Server gởi về.
* public OutputStream **getOutputStream()** :  Phương thức này trả về OutputStream nối với Socket. Chương trình Client dùng OutputStream này để gởi dữ liệu cho Server.
* public close() :  Phương thức này sẽ đóng Socket lại, giải phóng kênh ảo, xóa nối kết giữa Client và  
  Server.

Code của chương trình Client như sau:

package com.inpg.tcp;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

public class EchoChatClient {

    public final static String SERVER\_IP = "127.0.0.1";

    public final static int SERVER\_PORT = 7;

    public static void main(String[] args) throws IOException, InterruptedException {

        Socket socket = null;

        try {

            socket = new Socket(SERVER\_IP, SERVER\_PORT); // Connect to server

            System.out.println("Connected: " + socket);

            InputStream is = socket.getInputStream();

            OutputStream os = socket.getOutputStream();

            for (int i = '0'; i <= '9'; i++) {

                os.write(i); // Send each number to the server

                int ch = is.read(); // Waiting for results from server

                System.out.print((char) ch + " "); // Display the results received from the server

                Thread.sleep(200);

            }

        } catch (IOException ie) {

            System.out.println("Can't connect to server");

        } finally {

            if (socket != null) {

                socket.close();

            }

        }

    }

}

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

Một số phương thức cần thiết để xây dụng các chương trình Server sử dụng socket ở chế độ có nối kết.:

* 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.
* public Socket **accept()**: Phương thức này lắng nghe yêu cầu nối kết 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 nối kết của client gởi đến. Khi có yêu cầu nối kết của Client gởi đến, nó sẽ chấp nhận yêu cầu nối kết, trả về  
  một Socket là một đầu của kênh giao tiếp ảo giữa Server và Client yêu cầu nối kết.
* public InputStream **getInputStream()** : Phương thức này trả về InputStream nối với Socket. Chương trình Server dùng InputStream này để nhận dữ liệu từ Client gởi đến.
* public OutputStream **getOutputStream()** :  Phương thức này trả về OutputStream nối với Socket. Chương trình Server dùng OutputStream này để trả dữ liệu cho Client.
* public close() :  Phương thức này sẽ đóng Socket lại, giải phóng kênh ảo, xóa nối kết giữa Client và  
  Server.

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 nối kết. Các yêu cầu nối kết 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 nối kết và phục vụ nhiều Client cùng lúc.

#### Code của chương trình Server theo chế độ tuần tự như sau:

package com.inpg.tcp;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class EchoChatSingleServer {

    public final static int SERVER\_PORT = 7;

    public static void main(String[] args) throws IOException {

        ServerSocket serverSocket = null;

        try {

            System.out.println("Binding to port " + SERVER\_PORT + ", please wait  ...");

            serverSocket = new ServerSocket(SERVER\_PORT);

            System.out.println("Server started: " + serverSocket);

            System.out.println("Waiting for a client ...");

            while (true) {

                try {

                    Socket socket = serverSocket.accept();

                    System.out.println("Client accepted: " + socket);

                    OutputStream os = socket.getOutputStream();

                    InputStream is = socket.getInputStream();

                    int ch = 0;

                    while (true) {

                        ch = is.read(); // Receive data from client

                        if (ch == -1) {

                            break;

                        }

                        os.write(ch); // Send the results to client

                    }

                    socket.close();

                } catch (IOException e) {

                    System.err.println(" Connection Error: " + e);

                }

            }

        } catch (IOException e1) {

            e1.printStackTrace();

        } finally {

            if (serverSocket != null) {

                serverSocket.close();

            }

        }

    }

}

#### Code của chương trình Server theo chế độ song song như sau:

#### ****EchoChatMultiServer.java****

package com.inpg.tcp;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class EchoChatMultiServer {

    public static final int NUM\_OF\_THREAD = 4;

    public final static int SERVER\_PORT = 7;

    public static void main(String[] args) throws IOException {

        ExecutorService executor = Executors.newFixedThreadPool(NUM\_OF\_THREAD);

        ServerSocket serverSocket = null;

        try {

            System.out.println("Binding to port " + SERVER\_PORT + ", please wait  ...");

            serverSocket = new ServerSocket(SERVER\_PORT);

            System.out.println("Server started: " + serverSocket);

            System.out.println("Waiting for a client ...");

            while (true) {

                try {

                    Socket socket = serverSocket.accept();

                    System.out.println("Client accepted: " + socket);

                    WorkerThread handler = new WorkerThread(socket);

                    executor.execute(handler);

                } catch (IOException e) {

                    System.err.println(" Connection Error: " + e);

                }

            }

        } catch (IOException e1) {

            e1.printStackTrace();

        } finally {

            if (serverSocket != null) {

                serverSocket.close();

            }

        }

    }

}

#### ****WorkerThread.java****

package com.inpg.tcp;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.Socket;

public class WorkerThread extends Thread {

    private Socket socket;

    public WorkerThread(Socket socket) {

        this.socket = socket;

    }

    public void run() {

        System.out.println("Processing: " + socket);

        try {

            OutputStream os = socket.getOutputStream();

            InputStream is = socket.getInputStream();

            while (true) {

                int ch = is.read(); // Receive data from client

                if (ch == -1) {

                    break;

                }

                os.write(ch); // Send the results to client

            }

        } catch (IOException e) {

            System.err.println("Request Processing Error: " + e);

        } System.out.println("Complete processing: " + socket);

}

}

#### 3.Phân tích thiết kế

#### *\*Hình ảnh giao diên:*

#### 

#### *\*Khi server không nhận file:*

#### 

#### *\*Khi server nhận được file:*

#### 

#### KẾT LUẬN

Truyền file qua mạng là một ứng dụng phổ biến trên mạng LAN và INTERNET .Truyền file qua mạng đựa trên socket TCP là một phương thức truyền file có độ tin cậy cao bởi vì trước khi truyền nó cần thiết lập thành công kênh truyền dữ liệu.

Không phải là phương thức thay thế hoàn toàn những phương thức truyền file khác mà ta đã từng sử dụng.Bản chất của phương thức truyền file dựa vào socket TcP là nhằm tang thêm hiệu suất làm việc.Đề tài “Tìm hiểu mô hình lập trình mạng Socket trong Java. Sử dụng để viết chương trình truyền file đơn giữa 2 máy” đã đạt được kết quả nhất định.

Tuy nhiên,do hạn chế về thời gian và trình độ nên nhiều tính năng của chương trình chưa được hoàn thiện.Trong thời gian tới,chương trình sẽ được hoàn thiện theo hướng bổ sung các chức năng cho phù hợp yêu cầu đặc thù của truyền file qua mạng,có thể áp dụng vào thực tế cuộc sống.

\*\*\* THE END\*\*\*