# CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH CLIENT-SERVER SỬ DỤNG GIAO THỨC TCP

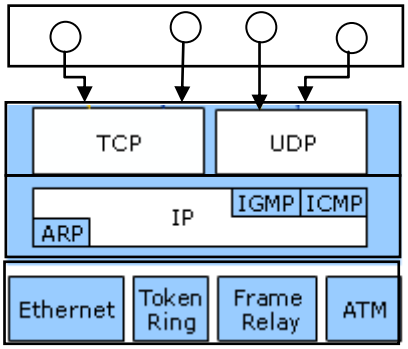
## I. Giới thiệu

Sau khi nghiên cứu các mô hình phân tầng mạng, quá trình biến đổi dữ liệu để truyền, nhận dữ liệu cũng như mô hình client-server. Chúng ta đã sẵn sàng cho việc xây dựng các dịch vụ ở tầng ứng dụng theo mô hình client-server.

Trong chương 1, chúng ta đã nghiên cứu về mô hình TCP/IP, một mô hình thực tế đang được sử dụng rất phổ biến hiện nay. Mô hình TCP/IP được xây dựng bao gồm tập hợp các giao thức (còn được gọi là các dịch vụ) được phân bố ở các tầng, phục vụ cho quá trình biến đổi và truyền, nhận dữ liệu. Để dữ liệu có thể được truyền đi trên mạng, dữ liệu được đưa vào từ người dùng phải được chia nhỏ, biến đổi qua nhiều bước thành định dạng có thể truyền được trên các phương tiện truyền.

Giả sử hệ thống mà chúng ta lập trình trên đó không hỗ trợ bất cứ giao thức (hay dịch vụ) nào phục vụ quá trình biến đổi và truyền, nhận dữ liệu thì người lập trình bắt buộc phải viết chương trình thực hiện tất cả các việc: tiếp nhận dữ liệu từ người dùng đưa vào, chia nhỏ dữ liệu, thêm các thông tin điều khiển phục vụ việc truyền dữ liệu, chuyển đổi dữ liệu sang dạng có thể truyền được trên các phương tiện truyền (như tín hiệu điện, sóng, ánh sáng). Nếu vậy, người lập trình sẽ rất khó khăn tạo ra được một dịch vụ theo mô hình client – server.

Tuy nhiên, rất may là mô hình TCP/IP đã cung cấp sẵn cho chúng ta gần như tất cả các giao thức (hay dịch vụ) ở tất cả các tầng. Người lập trình chỉ cần xây dựng các dịch vụ ở tầng ứng dụng, sử dụng các giao thức (hay dịch vụ) ở các tầng dưới là có thể truyền và nhận dữ liệu qua mạng. Để thực hiện việc này, người lập trình chỉ cần viết chương trình cung cấp giao diện để nhập hoặc nhận dữ liệu, sau đó nối với một trong hai giao thức (thực tế là hai dịch vụ) TCP hoặc UDP của tầng Vận chuyển để các giao thức này đảm nhận các việc còn lại của quá trình gửi và nhận dữ liệu.




*Hình 20. Chương trình ứng dụng sử dụng giao thức TCP và UDP*

Có nhiều dịch vụ (giao thức) ở tầng ứng dụng đã được phát triển sử dụng giao thức TCP của tầng Vận chuyển như FTP, Telnet, HTTP, NetBIOS.

Trong chương này, chúng ta sẽ nghiên cứu kỹ thuật xây dựng ứng dụng mô hình client-server bằng cách sử dụng giao thức TCP. Hay nói cách khác, nghiên cứu cách thức xây dựng chương trình kết nối với giao thức TCP của mô hình TCP/IP.

## II. Tổng quan về giao thức TCP

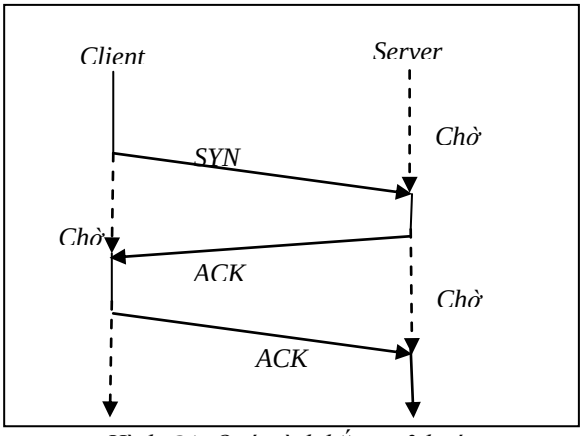
Trong chương 1 chúng ta đã đề cập đến giao thức TCP ở tầng Vận chuyển của bộ giao thức TCP/IP. Trong chương này chúng ta sẽ tìm hiểu sâu về giao thức này cũng như cách lập trình ứng dụng theo mô hình client-server sử dụng giao thức này.

Giao thức TCP là giao thức truyền thông có kết nối (connection oriented) và tin cậy (reliable).

Truyền thông có kết nối có nghĩa là trước khi truyền dữ liệu, giao thức TCP gửi và giao thức TCP nhận phải thiết lập kênh truyền trước khi gửi/nhận dữ liệu và, trong suốt quá trình trao đổi dữ liệu hai giao thức gửi/nhận phải duy trì kênh truyền đã được thiết lập.

Quá trình thiết lập kênh truyền được thực hiện cụ thể bởi một quá trình gọi là quá trình bắt tay ba bước (three-way handshake) như sau:

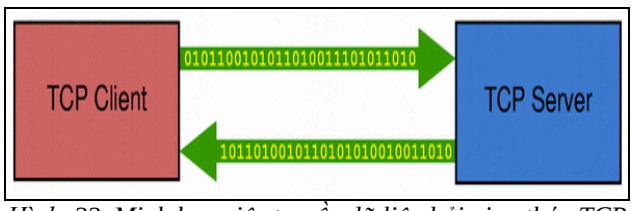
1. Trước hết, giao thức TCP phía máy gửi phải yêu cầu thiết lập một kênh truyền bằng cách gửi một phân đoạn (segment) gọi là SYN đến giao thức TCP phía máy nhận.
2. Giao thức TCP phía máy nhận trả lại một phân đoạn (segment) gọi là ACK để xác nhận đã nhận được thành công.
3. Giao thức TCP phía máy gửi tiếp tục gửi một phân đoạn ACK để xác nhận và sau đó là quá trình gửi dữ liệu.




*Hình 21. Quá trình bắt tay 3 bước*

Truyền thông tin cậy có nghĩa là nút gửi biết được gói tin đã được gửi đến đích hay không. Nếu không đến được, nút gửi phải gửi lại gói tin. Nếu gói tin đã được gửi thành công, nút gửi gửi gói tin tiếp theo. Cụ thể của quá trình này được thực hiện bằng cách mỗi gói tin được đánh một số thứ tự. Mỗi khi nhận được gói tin, nút nhận phải gửi một phản hồi xác nhận (acknowledgment) cho nút gửi để thông báo đã nhận được thành công gói tin.

Giao thức TCP sử dụng truyền thông dạng luồng dữ liệu hay dãy các bytes (byte-stream) liên tục trên kênh truyền, được mô tả như hình vẽ bên dưới:




*Hình 22. Minh họa việc truyền dữ liệu bởi giao thức TCP*

Mỗi gói tin TCP (hay còn gọi là segment) gồm một số thông tin điều khiển ở phần đầu được mô tả ở bảng dưới đây.

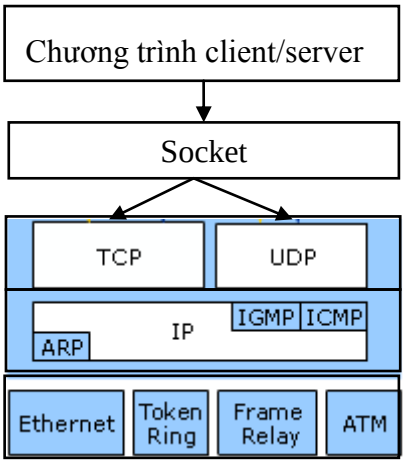
| Thông tin điều khiển | Mô tả |
|---|---|
| Số cổng nguồn (Source Port) | Số hiệu cổng TCP của tiến trình gửi |

| | |
|---|---|
| Số cổng đích (Destination Port) | Số hiệu cổng TCP của tiến trình nhận |
| Số hiệu gói tin (Sequence Number) | Số thứ tự của byte đầu tiên của gói tin TCP |
| Số hiệu xác nhận (Acknowledgment Number) | Số thự của byte đầu tiên của gói tin TCP mà giao thức TCP gửi chờ nhận từ phía nhận. |
| Kích thước bộ đệm (Window) | Kích thước hiện tại của bộ đệm TCP phía máy gửi dùng để chứa gói tin TCP nhận được. |
| Thông tin kiểm tra lỗi (TCP Checksum) | Dùng để kiểm tra tính toàn vẹn của phần đầu và dữ liệu của gói tin TCP. |

Để kết thúc kết nối, giao thức TCP cũng sử dụng cơ chế bắt tay 3 bước. Điều này đảm bảo cả hai giao thức TCP (gửi và nhận) đã kết thúc truyền dữ liệu và tất cả dữ liệu đã được gửi và nhận thành công.

## III. Khái niệm Socket

Như ở trên đã đề cập, để xây dựng một chương trình theo mô hình client-server, cho phép truyền nhận dữ liệu qua mạng, người lập trình chỉ cần viết chương trình ở tầng ứng dụng và tìm cách giao tiếp với một trong hai giao thức (dịch vụ) ở tầng Vận chuyển là TCP hoặc UDP. Để làm được việc này, các hệ điều hành cung cấp một phương tiện cho phép ứng dụng người dùng giao tiếp được với các giao thức TCP và UDP một cách dễ dàng. Phương tiện đó được gọi là Socket, được biểu diễn như hình vẽ bên dưới.



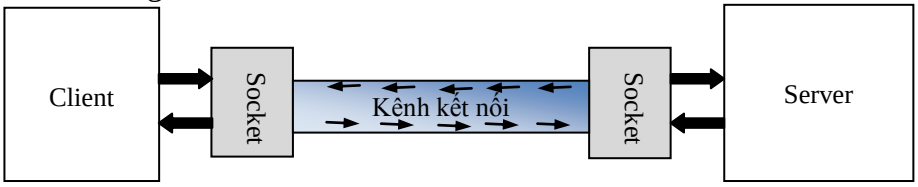

*Hình 23. Mô tả Socket trong mô hình TCP/IP*

Socket được xem như điểm giao tiếp của ứng dụng với mạng, do đó chương trình client/server sẽ thực hiện hai thao tác cơ bản sau:

- Để truyền dữ liệu, chương trình chỉ cần gửi dữ liệu vào socket
- Để nhận dữ liệu, chương trình chỉ cần đọc dữ liệu từ socket

Có hai loại socket: socket nối với giao thức TCP còn được gọi là *TCP socket*, và tương tự như vậy socket nối với giao thức UDP được gọi là *UDP socket*.

Một TCP socket được xem như là điểm cuối của kênh kết nối hai chiều liên kết giữa hai chương trình trên mạng.



*Hình 24. Socket trong mô hình ứng dụng client-server*

Có thể nói một kênh kết nối giữa hai chương trình được thiết lập nhờ vào hai socket được gắn ở hai chương trình. Như hình vẽ trên, mỗi chương trình client và server đều có một socket được gắn vào nó để có thể truyền và nhận dữ liệu.

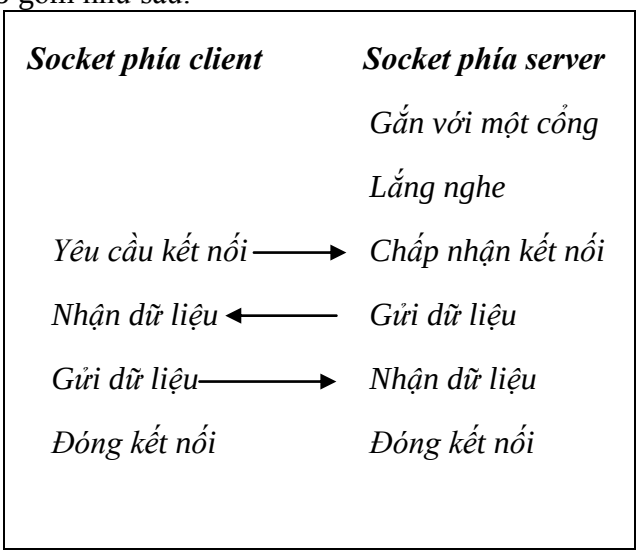
# IV. Xây dựng chương trình Client-Server với giao thức TCP

## *IV.1. Sử dụng TCP Socket*

Như trình bày ở mục trên, việc xây dựng chương trình client-server sử dụng giao thức TCP của tầng Vận chuyển được thực hiện thông qua TCP socket. Do vậy, người lập trình chỉ cần viết mã lệnh để đọc và ghi dữ liệu giữa chương trình và TCP socket, sau đó TCP socket sẽ có nhiệm vụ liên lạc và gửi, nhận dữ liệu với TCP socket phía đầu còn lại.

Do đó, các hoạt động của TCP socket sẽ bao gồm như sau:

- **TCP Socket phía server:**
  - Lắng nghe yêu cầu kết nối từ socket phía client
  - Chấp nhận kết nối từ socket phía client
  - Gửi dữ liệu
  - Nhận dữ liệu
  - Đóng kết nối
- **TCP Socket phía client:**
  - Yêu cầu kết nối với socket phía server
  - Gửi dữ liệu
  - Nhận dữ liệu
  - Đóng kết nối




Để TCP socket phía client có thể liên lạc được với TCP socket phía server thì TCP socket phía server phải được gắn với một cổng (port). Sau khi TCP socket phía server được gắn một cổng, nó sẽ chờ và lắng nghe yêu cầu tạo kênh kết nối từ TCP socket phía client gửi đến.

Thực tế, socket được tồn tại dưới dạng các thư viện API do các hệ điều hành hoặc các ngôn ngữ lập trình cung cấp cho phép người lập trình sử dụng chúng để xây dựng chương trình client-server.

Chẳng hạn, hệ điều hành UNIX cung cấp thư viện có tên Berkeley socket dưới dạng các API, trong khi đó hệ điều hành Windows cung cấp thư viện API có tên Windows Socket (còn gọi là winsock). Trong ngôn ngữ lập trình Java có gói *java.net*, trong khi đó ngôn ngữ lập trình C có thư viện *sys/socket.h*.

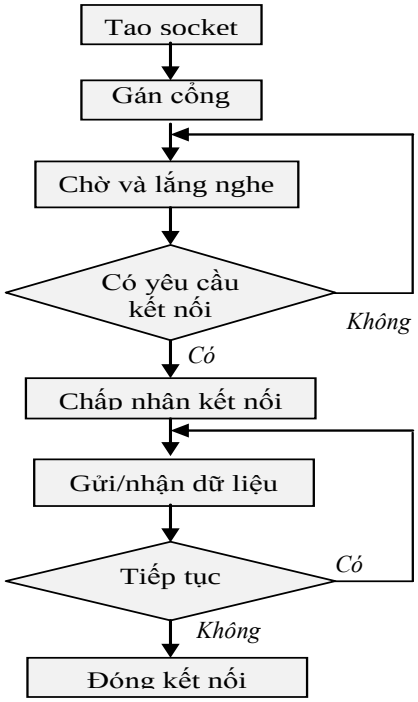
Người lập trình chỉ cần sử dụng các thư viện chương trình của Socket được cung cấp bởi hệ điều hành hoặc ngôn ngữ lập trình để xây dựng chương trình theo mô hình client-server.

## *IV.2. Các bước và thuật toán xây dựng chương trình Client-Server*

### a. Các bước xây dựng chương trình server

Chương trình server được xây dựng gồm các bước sau:

1. Tạo TCP socket
2. Gắn TCP socket với một số hiệu cổng
3. Chờ và lắng nghe yêu cầu kết nối từ chương trình client
4. Nếu có yêu cầu kết nối, chấp nhận kết nối từ chương trình client
5. Sau khi kênh kết nối được thiết lập, trao đổi dữ liệu với chương trình client
6. Đóng kết nối

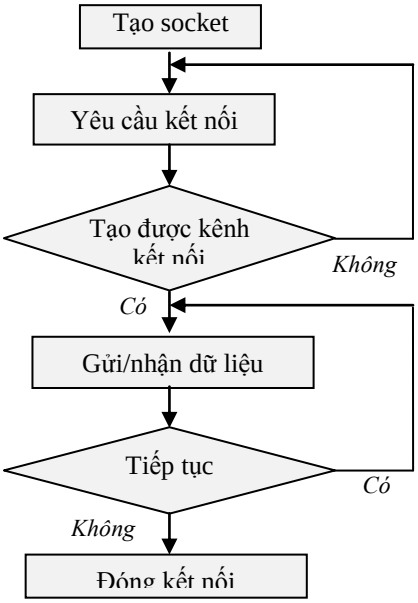



*Hình 25. Thuật toán xây dựng chương trình Server*

## b. Các bước xây dựng chương trình client

Chương trình client được xây dựng gồm các bước sau:

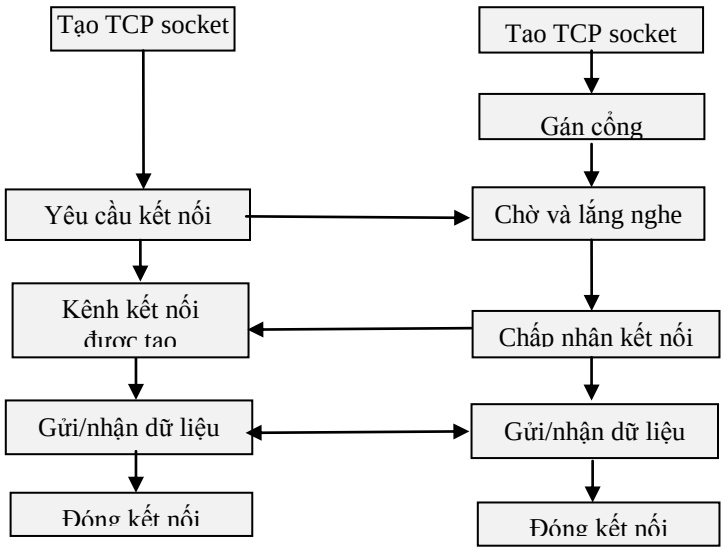
1. Tạo TCP socket
2. Yêu cầu kết nối đến server. Nếu được server chấp nhận, một kênh kết nối được thiết lập
3. Trao đổi dữ liệu với chương trình server
4. Đóng kết nối




*Hình 26. Thuật toán xây dựng chương trình client*

## c. Thuật toán xây dựng chương trình client – server tổng quát

Từ các bước xây dựng chương trình client và server ở trên, quá trình xây dựng một ứng dụng theo mô hình client-server được mô tả bằng thuật toán ở sơ đồ khối như sau:




*Hình 27. Thuật toán xây dựng ứng dụng client-server tổng quuats*

Sau khi chương trình server tạo TCP socket và gắn một số hiệu cổng vào TCP socket, chương trình server bắt đầu chờ và lắng nghe yêu cầu kết nối từ chương trình client.

Do TCP là giao thức truyền thông có kết nối nên để trao đổi được dữ liệu với chương trình server, chương trình client phải tạo một kênh kết nối đến chương trình server. Chương trình client đầu tiên tạo một TCP socket, sau đó gửi yêu cầu kết nối đến chương trình server đang chạy tại một địa chỉ IP và số hiệu cổng đã xác định.

Chương trình server nhận được yêu cầu kết nối gửi đến, nếu không có lỗi sẽ chấp nhận kết nối. Ngay khi chương trình server chấp nhận kết nối, một kênh kết nối được thiết lập để cho phép chương trình client và server trao đổi dữ liệu với nhau.

## V. Triển khai xây dựng chương trình client – server bằng ngôn ngữ Java

Hiện nay có nhiều nhiều ngôn ngữ lập trình cho phép xây dựng chương trình client-server, tuy nhiên trong cuốn sách này chúng tôi sử dụng ngôn ngữ lập trình Java bởi ngôn ngữ này có nhiều tính ưu việt hỗ trợ cho lập trình mạng.

Ngôn ngữ lập trình Java cung cấp gói *java.net* cho phép xây dựng ứng dụng theo mô hình client – server. Do đó, ở phần khai báo sử dụng gói của chương trình client và server cần phải khai báo câu lệnh sử dụng gói này như sau:

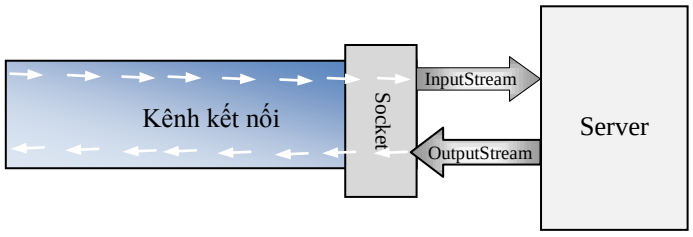
*import java.net.*;*

- **Viết chương trình server**

Đầu tiên, chúng ta triển khai viết chương trình server dựa theo các bước ở mụcCHƯƠNG 3. IV.2.a. của chương này. Tương ứng mỗi bước (hoặc nhiều bước) sẽ là một hoặc một số lệnh trong ngôn ngữ Java được mô tả như sau:

| Các bước thuật toán | Mã lệnh trong ngôn ngữ Java |
|---|---|
| 1. Tạo TCP socket<br>2. Gắn TCP socket với một số hiệu cổng | Cả hai bước tạo socket và gắn số hiệu cổng cho socket tương ứng với một câu lệnh java dưới đây.<br>*ServerSocket server = new ServerSocket(port);* |
| 3. Chờ và lắng nghe yêu cầu kết nối từ chương trình client<br>4. Nếu có yêu cầu kết nối, chấp nhận kết nối từ chương trình client | Các hoạt động chờ, lắng nghe và chấp nhận kết nối được thực hiện bởi phương thức accept của lớp ServerSocket.<br>*Socket socket = server.accept();*<br>Khi hàm accept được thực hiện, một đối tượng socket được tạo ra. Điều này có nghĩa là kênh kết nối được thiết lập bởi hai socket (phía server và phía client). |
| 5. Trao đổi dữ liệu với chương trình server | Giao thức TCP sử dụng truyền thông dạng luồng dữ liệu hay dãy các bytes (byte-stream) liên tục trên kênh truyền, do đó chương trình cũng phải truyền dữ liệu đến socket dưới dạng luồng dữ liệu.<br>Muốn truyền dữ liệu đến socket, chương trình chỉ cần tạo một luồng (stream) để ghi dữ liệu lên socket, tương ứng câu lệnh java như sau:<br>*OutputStream stream_out = socket.getOutputStream();*<br>Tương tự, muốn nhận dữ liệu từ socket, chương trình chỉ cần tạo một luồng để đọc dữ liệu từ socket, tương ứng câu lệnh java như sau.<br>*InputStream stream_in = socket.getInputStream();*<br>Như vậy, tương ứng với việc truyền dữ liệu từ chương trình đến socket là câu lệnh ghi dữ liệu từ chương trình vào stream_out. Ngược lại, tương ứng với việc nhận dữ liệu từ socket vào chương trình là câu lệnh đọc dữ liệu từ stream_in vào chương trình. |
| 6. Đóng kết nối | *socket.close();* |

Như bảng mô tả ở trên, lớp Socket trong Java cung cấp các hàm cho phép tạo ra các luồng ghi và đọc (OutputStream và InputStream) cho phép chương trình ghi và đọc dữ liệu lên Socket một các dễ dàng. Socket sẽ chịu trách nhiệm liên lạc với socket ở đầu còn lại để gửi và nhận dữ liệu.




*Hình 28. Chương trình Server gửi nhận dữ liệu nhờ vào các luồng dữ liệu*

Ví dụ 1: Chương trình server cơ bản được viết bằng Java như sau:

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

class TCPServer
{
            String text;
             ServerSocket server;

             public TCPServer()
             {
           try{
                 server = new ServerSocket(7000);

           System.out.println("Server is starting...");

          Socket socket = server.accept();

           BufferedReader stream_in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
           DataOutputStream stream_out = new DataOutputStream(socket.getOutputStream());

           text = stream_in.readLine();
           System.out.println("Received: " + text);

           text = text.toUpperCase() + '\n';
          stream_out.writeBytes(text);
           }
           catch(Exception e) {e.printStackTrace();}
        }

  public static void main(String argv[])
      {
      new TCPServer();
      }
}
```

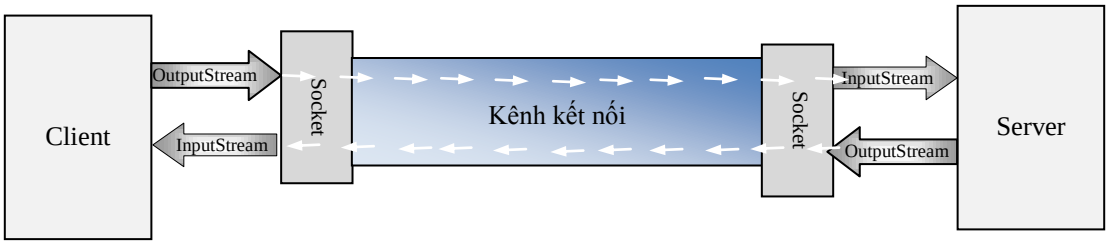
- **Viết chương trình client**

Sau khi viết chương trình server, bây giờ chúng ta tiến hành viết chương trình client dựa vào các bước và thuật toán được mô tả ở mục CHƯƠNG 3. IV.2.b. của chương này. Tương tự như trình tự xây dựng chương trình server, dưới đây là bảng tương ứng giữa thuật toán và mã lệnh trong ngôn ngữ Java để xây dựng chương trình client.

| Các bước thuật toán | Mã lệnh trong ngôn ngữ Java |
|---|---|
| 1. Tạo TCP socket<br>2. Yêu cầu kết nối đến server. | Cả hai bước tạo socket và yêu cầu kết nối đến server được thực hiện bởi một câu lệnh Java sau.<br>*Socket socket = new Socket(IP, port);* |
| 3. Trao đổi dữ liệu với chương trình server | Tương tự như chương trình server, để truyền và nhận dữ liệu giữa chương trình client với socket, chương trình client cần phải tạo ra các luồng dữ liệu để ghi và đọc dữ liệu với socket. |
| 4. Đóng kết nối | *socket.close();* |

Giống như chương trình server, chương trình client chỉ cần truyền và nhận dữ liệu với socket thông qua việc đọc và ghi dữ liệu với các luồng InputStream và OutputStream. Socket ở phía

client sẽ chịu trách nhiệm liên lạc với socket ở phía server để gửi và nhận dữ liệu đến chương trình server.




*Hình 29. Chương trình Client gửi nhận dữ liệu với chương trình Server*

Ví dụ 2: Chương trình client cơ bản viết bằng Java như sau:

```
import java.io.*;
import java.net.Socket;
import java.util.*;
/*Chương trình client:
-  cho phép người dùng nhập chuỗi từ  bàn phím để gửi đến chương trình Server
- nhận chuỗi từ server gửi về và hiển thị lên màn hình
*/
public class Client {
     Socket socket ;
     BufferedReader stream_in ;
     PrintWriter stream_out;

     public Client()
     {

     try{
       Socket socket = new Socket("localhost", 7000); //gửi yêu cầu kết nối đến server
       BufferedReader stream_in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
       PrintWriter stream_out = new PrintWriter(socket.getOutputStream(), true);

       //Tạo luồng nhận dữ liệu từ bàn phím
        Scanner kb = new Scanner(System.in);

      //Nhập chuỗi từ bàn phím
       System.out.print("Enter a string: ");
       String msg=kb.nextLine();

     //Gửi dữ liệu đến server
      stream_out.println(msg);
      stream_out.flush();

     //nhận dữ liệu từ server và hiển thị lên màn hình
      String st = stream_in.readLine();
      System.out.println(st);
     kb = kb.reset();

     socket.close();
    }catch(Exception e){}
  }

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

```
    new Client();
    }
}
```

Hai ví dụ trên minh họa cách thức xây dựng hai chương trình (client và server) trao đổi dữ liệu với nhau. Trong đó, chương trình server được gán số hiệu cổng 7000 để chờ và lắng nghe yêu cầu kết nối từ chương trình client. Chương trình client gửi yêu cầu kết nối đến chương trình server đang chạy trên cùng một máy với nó (có địa chỉ máy được ký hiệu "localhost") và ở số hiệu cổng 7000 bằng câu lệnh:

```
Socket socket = new Socket("localhost", 7000);
```

Ngay khi câu lệnh trên được thực hiện ở chương trình client thì chương trình server thực hiện câu lệnh bên dưới để chấp nhận kết nối:

```
Socket socket = server.accept();
```

Ngay khi câu lệnh trên ở phía server được thực hiện, một kênh kết nối được thiết lập giữa chương trình client và server nhờ vào hai socket ở hai đầu.

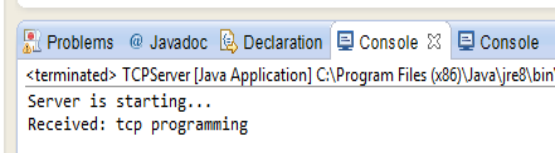
Dựa vào kênh kết nối đã được thiết lập, chương trình client tạo luồng để ghi dữ liệu lên socket có tên stream_out để gửi một chuỗi msg được nhập từ bàn phím đến chương trình server bằng các câu lệnh:

```
PrintWriter stream_out = new PrintWriter(socket.getOutputStream(), true);
String msg=kb.nextLine();
stream_out.println(msg);
```

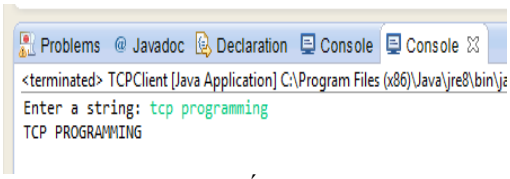
Sau khi chương trình client gửi chuỗi dữ liệu đến chương trình server, dữ liệu được chuyển đến socket phía server, chương trình server sẽ đọc dữ liệu từ socket vào chương trình bằng cách tạo luồng đọc dữ liệu từ socket có tên stream_in, chuyển đổi chuỗi nhận được thành chữ hoa rồi gửi ngược lại cho client bằng các lệnh sau:

```
BufferedReader stream_in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
DataOutputStream stream_out = new DataOutputStream(socket.getOutputStream());

text = stream_in.readLine();                //nhận dữ liệu gửi đến từ client
System.out.println("Received: " + text);   // hiển thị lên màn hình
text = text.toUpperCase() + '\n';           //chuyển đổi thành chuỗi chữ hoa
stream_out.writeBytes(text);                //gửi trả chuỗi lại cho client
```

Kết quả chạy các chương trình trên sẽ như sau:




*Hình 30. Kết quả chạy server*




*Hình 31. Kết quả chạy client*

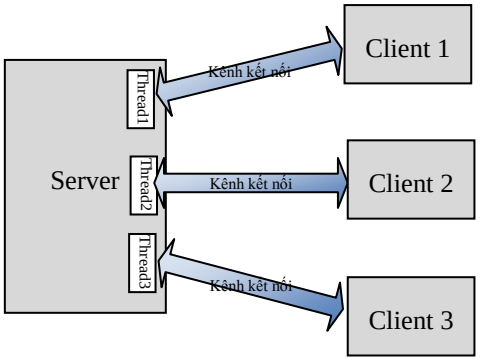
Chúng ta vừa tìm hiểu cách xây dựng chương trình client và chương trình server đơn giản bằng ngôn ngữ Java. Qua hai ví dụ trên, có thể nhận thấy rằng việc viết chương trình theo mô hình client và server khá đơn giản.

Tuy nhiên, hạn chế của chương trình trên là tại một thời điểm chương trình server chỉ kết nối và phục vụ cho một client, server không thể kết nối và phục vụ nhiều client bởi vì giao thức TCP chỉ cho phép truyền thông điểm-điểm.

- **Viết chương trình server hoạt động đa tuyến (multi-threadings)**

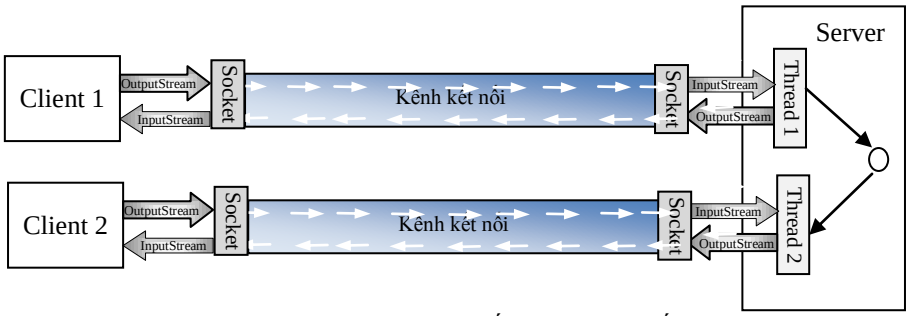
Để cho phép chương trình server kết nối và phục vụ được nhiều chương trình client tại một thời điểm, chúng ta cần phải xây dựng chương trình server là chương trình đa tuyến. Trong đó, mỗi tuyến (thread) ở chương trình server sẽ giữ một kết nối với một chương trình client và đảm nhận việc truyền nhận dữ liệu với client đó.

Khi có một chương trình client gửi yêu cầu kết nối đến chương trình server thì chương trình server phải tạo ra một tuyến (thread) để kết nối và trao đổi dữ liệu với client đó.




*Hình 32. Minh họa server đa tuyến*

Để mỗi đối tượng thread có thể giữ một kênh kết nối với một client, thì một đối tượng thread được tạo ra phải gắn với một socket như đầu cuối của kênh kết nối giữa nó với client.




*Hình 33. Mô hình chi tiết Server đa tuyến*

Đoạn mã lệnh chương trình server mô tả quá trình tạo một đối tượng thread để giữ kênh kết nối và liên lạc với client khi có yêu cầu kết nối được gửi đến như sau:

```
Server
...
socket = server.accept();
thread[i] = new ThreadHandler(socket);
....
```

Ở đoạn mã trên, lớp ThreadHandler được định nghĩa dẫn xuất từ lớp Thread, phải có 3 thuộc tính quan trọng gồm: một đối tượng của lớp Socket, một đối tượng của lớp InputStream và một đối tượng của lớp OutputStream. Ba thuộc tính này cho phép một thread được tạo ra có

khả năng kết nối với một client (nhờ hai socket ở hai đầu) và có khả năng gửi, nhận dữ liệu nhờ các đối tượng OutputStream và InputStream.

Lớp ThreadHandler có thể định nghĩa như sau:

```
class ThreadHandler extends Thread
{
   Socket socket;
   InputStream stream_in;
   OutputStream stream_out;
   public ThreadHandler(Socket s)
    {
    socket = s;
    stream_in  = new DataInputStream(socket.getInputStream());
    stream_out = new DataOutputStream(socket.getOutputStream());
    }
  public void run()
   {
   /*trao đổi dữ liệu với client thông qua các đối tượng stream_in và
     stream_out được thực hiện ở đây */
    }
}
```

Mỗi thread hoạt động như một chương trình server, nó giao tiếp và trao đổi dữ liệu với socket được gắn với nó thông qua các luồng đọc và ghi dữ liệu (InputStream và OutputStream) để có thể trao đổi với client tương ứng.

Bây giờ chúng ta phát triển chương trình server ở ví dụ 2, cho phép làm viêc với nhiều client cùng một thời điểm.

Ví dụ 3:

```
import java.net.*;
import java.io.*;
import java.util.*;
public class MultiThread_TCPServer
{
 private static ServerSocket  ser;

 public MultiThread_TCPServer()
  {
 try{
   ser = new ServerSocket(9891);
   System.out.println("Server is working...");
   while (true)
    {
     ThreadHandler s = new ThreadHandler(ser.accept());
     s.start();
    }
  }catch(Exception e){}
 }
 public static void main(String[] args)
     {
     new MultiThread_TCPServer();
     }
}
```

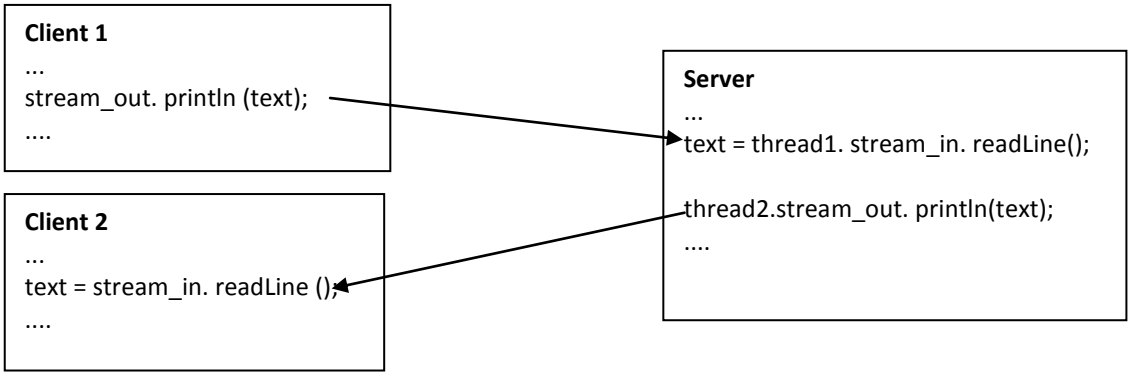
```
class ThreadHandler extends Thread
{
            private PrintWriter out;
   private BufferedReader bf;
   private Socket sk;
    public ThreadHandler(Socket soc)
            {
            try{
              sk=soc;
               bf = new BufferedReader(new InputStreamReader(sk.getInputStream()));
               out = new PrintWriter(sk.getOutputStream(), true);
            }catch(Exception e){}

            }

     public void run()
      {
     try{
        while (true)
          {
          String s = bf.readLine();
           s=s.toUpperCase() + '\n';
           out.println(s);
           out.flush();
          }
        }catch(Exception e){}
      }
 }
}
```

Chương trình trên cho phép một server có thể kết nối và phục vụ nhiều client tại một thời điểm, tuy nhiên chưa cho phép hai client trao đổi dữ liệu được với nhau. Như đã trình bày ở chương 2, hai client không thể trao đổi dữ liệu trực tiếp với nhau, tất cả mọi mối liên lạc giữa các client đều thông qua chương trình server.

Dưới đây là thuật toán cho phép hai client trao đổi dữ liệu với nhau. Giả sử client 1 muốn trao đổi dữ liệu với client 2, quá trình trao đổi được minh họa ở Hình 33 như sau:

- Client 1 gửi dữ liệu đến server thông qua đối tượng thread 1
- Chương trình server đọc dữ liệu từ đối tượng thread 1
- Chương trình server nhờ đối tượng thread 2 gửi dữ liệu nhận được đến client 2

Các đoạn mã lệnh tương ứng cho quá trình *client 1* gửi dữ liệu cho *client 2* như sau:



## VI. Bài tập

1. Hãy chạy các chương trình ở ví dụ 1 (chương trình server) và ví dụ 2 (chương trình client) của chương này.
2. Hãy chạy chương trình ở ví dụ 3 (chương trình server) và ở ví dụ 2 (chương trình client).
3. Hãy phát triển chương trình ở ví dụ 2, cho phép người dùng gửi và nhận chuỗi nhiều lần (gợi ý: thêm câu lệnh lặp).
4. Từ chương trình ở ví dụ 3, hãy viết chương trình chat cho phép các client gửi dữ liệu nhập từ bàn phím qua lại.
5. Phát triển bài tập 4, chương trình server có kết nối CSDL để quản lý tài khoản người dùng.
6. Hãy viết chương trình client cho phép người dùng nhập vào 2 số thực và một phép toán (+, -, *, /) rồi gửi đến chương trình Server. Chương trình Server (đa tuyến) thực hiện tính toán kết quả dựa vào phép toán tương ứng và trả kết quả cho chương trình Client. Mỗi lần tính toán được, server đều lưu kết quả xuống CSDL.