Đồ án socket

Vu Cong Duy - 19120212

Nguyen Hoang Anh Kiet - 19120266

December 2020

Chapter 1

Socket

1.1 Lựa chọn giao thức tầng ứng dụng

Trước khi bắt đầu lập trình một chương trình ứng dụng socket mạng, việc quan trọng nhất chính là việc chọn giao thức tầng transport mà ứng dụng đó sẽ dùng.[1] Bởi đây là nơi sâu nhất mà một lập trình viên có thể can thiệp, giao thức tầng transport sẽ ảnh hưởng tới toàn bộ chương trình.

Có 2 giao thức phổ biến : UDP và TCP. Ta có thể thấy, việc truyền tải dữ liệu một trang web đòi hỏi phải có độ tin cậy cao. Vì vậy, giao thức truyền dữ liệu TCP sẽ là phù hợp hơn cho chương trình socket này. Thật vậy, giao thức HTTP (giao thức về truyền dữ liệu web) ở tầng application cũng đã chọn giao thức TCP để truyền tải dữ liệu trang web.

1.2 Mô hình triển khai

Về cơ bản, mọi chương trình sử dụng giao thức TCP đều có mô hình Client-Server triển khai như sau.

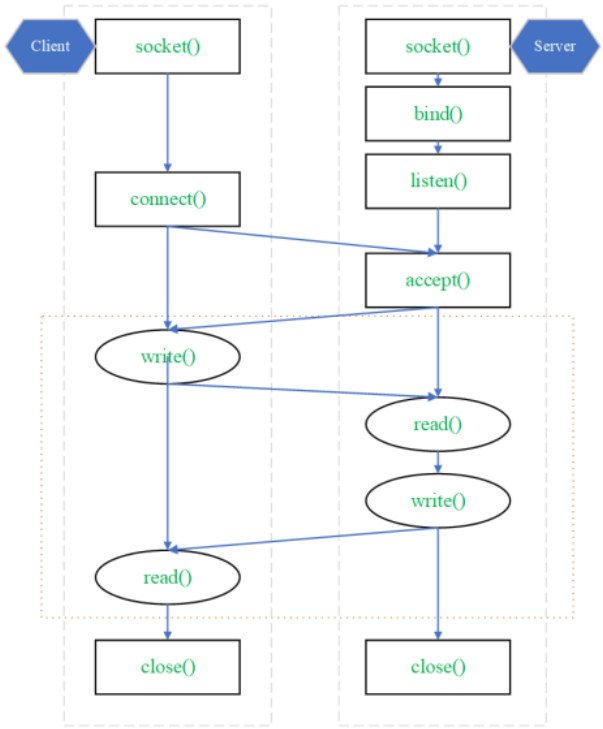

Figure 1.1: Mô hình Client-Server triển khai theo giao thức tcp

Trong đồ án này, chương trình Client chính là web browser đã được lập trình sẵn, vì vậy phần này sẽ được duyệt nhanh và chú trọng hơn vào phần lập trình socket Server.

1.2.1 Giai đoạn 1: Khởi tạo socket cho server

Với server:

-Server tạo socket bằng lệnh **socket()**. Với ngôn ngữ python, việc này thực hiện bằng hàm socket() trong thư viện socket (đã được import từ trước).Hàm socket còn có thể truyền tham số đầu vào như giao thức địa chỉ IP (IPv4 và IPv6) và giao thức tầng transport (UDP và TCP). Nếu không truyền tham số, hàm tự động tạo một socket có giao thức

IPv4/TCP. Những giá trị hằng cho các tham số trên được quy định như hình dưới.

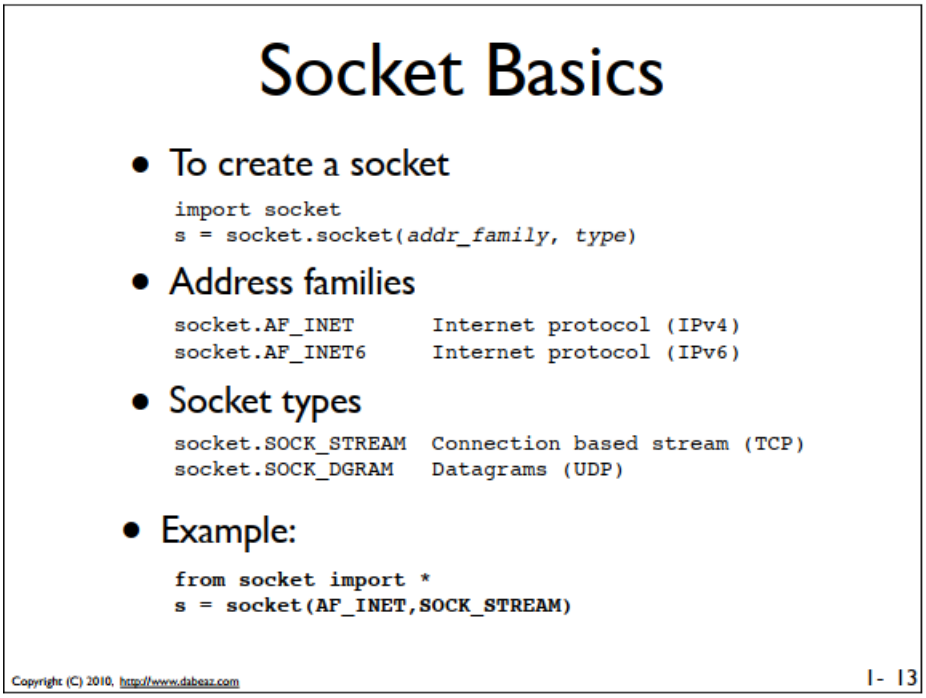

Figure 1.2: Giá trị các tham số cho hàm socket [2]

- Sau đó, server gán socket này một port (ở trường hợp này là port 80) và địa chỉ IP của server (ở đây là máy đang chạy chương trình). Trong ngôn ngữ python, hành động này được thực hiện bằng hàm **bind()**. Tham số truyền vào chính là một tuple biểu diễn cặp địa chỉ IP - port mà chương trình muốn gán cho socket. Địa chỉ IP được viết dưới kiểu dữ liệu chuỗi, trong khi port là kiểu dữ liệu số nguyên.

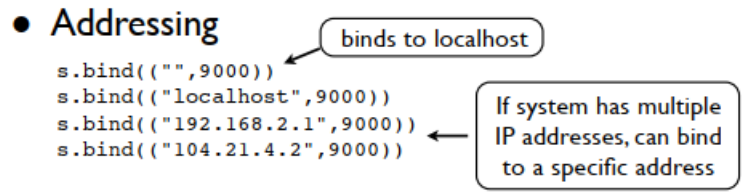

Figure 1.3: Minh họa cho hàm bind [2]

- Cuối cùng, cho sever bắt đầu lắng nghe yêu cầu bằng lệnh **listen()**. [2]Tham số đầu có kiểu dữ liệu số nguyên, phụ thuộc vào số kết nối trong hàng chờ được phép có (hoàn toàn không phụ thuộc vào số client tối đa).

```
hostname = socket.gethostname()
host = socket.gethostbyname(hostname)
#Set up socket and listen
server = socket.socket()
server.bind((host,80))
print("ip : ",host)      You, 5 hours ago • socket done
server.listen(1)
```

Figure 1.4: Các bước thiết lập socket Server hoàn chỉnh để sẵn sàng lắng nghe yêu cầu từ Client

1.2.2 Giai đoạn 2: Khởi tạo socket cho client

Việc khởi tạo socket cho client cũng có nhiều điểm tương đồng với tạo socket cho server.

- Client yêu cầu tạo một socket với lệnh **socket()**. Tương ứng với lệnh socket() của server, lệnh tạo socket từ client cũng yêu cầu phương thức IP và phương thức của tầng transport. Web browser sẽ mặc định dùng giao thức TCP. -Sau khi tạo socket, client sẽ kết nối socket vừa

tạo đến socket của server, với lệnh **connect()**. Lệnh này có tham số đầu vào tương đồng với lệnh **bind()** của server khi đều yêu cầu địa chỉ IP và port của socket server. Với web browser, các nhà phát triển đã lập trình tham số địa chỉ IP chính là địa chỉ IP (của tên miền nếu user nhập tên miền) nhập trên thanh truy cập, còn port mặc định là 80.

Ngay sau khi client thực hiện lệnh **connect()** thì tại server: -Server nhận được thông điệp yêu cầu kết nối từ lệnh connect() thì thực hiện lệnh chấp nhận bằng lệnh **accept()**, đồng thời cũng tạo ra một socket mới mà việc **truyền thông điệp response - request sẽ diễn ra trên socket này** (chứ không trên socket ban đầu mà server đã tạo!). Trong ngôn ngữ Python, method connect() sẽ trả giá trị là một tuple gồm socket mới và địa chỉ IP của client.

1.2.3 Giai đoạn 3: Trao đổi thông điệp giữa client và server

Hai thao tác cơ bản trong giai đoạn này là gửi và nhận thông tin. Trong ngôn ngữ python, các hàm gửi và nhận dưới dạng method của đối tượng socket lần lượt là :

send()/sendall() : biến đầu vào là dữ liệu được mã hóa nhị phân

recieve(): biến đầu vào là kích thước biến nhận dữ liệu, dữ liệu nhận được là dữ liệu nhị phân, muốn đọc hoặc xử lý thì phải mã hóa.

```
stream,addr=server.accept()

#receive request from client by new socket
request=stream.recv(10000)
#print(request)#debug
#Process the request
request=request.decode()
request_lines=request.splitlines()
print(request) #debug
```

Figure 1.5: Server thực hiện lệnh accecpt và recieve

Chi tiết phần trao đổi thông điệp giữa client và server sẽ được trình bày chi tiết ở Chương 2.

1.2.4 Giai đoạn 4: Kết thúc phiên làm việc

Sau khi trao đổi thông điệp kết thúc, server thực hiện đóng socket mà dùng để trao đổi thông tin với client và client cũng làm điều tương tự. Lưu ý : socket server đóng là socket phát sinh bởi hàm connect(), không phải là socket tạo từ đầu. Trong thực tế server luôn ở trong trạng thái hoạt động nên socket server tạo từ đầu sẽ không bao giờ đóng.

Ở ngôn ngữ python, socket được đóng bằng lệnh **close()**, không yêu cầu tham số đầu vào.

Chapter 2

Thông điệp HTTP

2.1 Cấu trúc request

Một request hoàn chỉnh có rất nhiều trường dữ liệu, nhưng về cơ bản, phần quan trọng nhất để một request có thể xử lý được là :

<tên method> <đường dẫn file> <tên phiên bản http>
Host : <địa chỉ IP của client>
. . .
Transfer-encoding:<cách thức encode>

<data>

```
GET / HTTP/1.1
Host: 192.168.12.2
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*
Purpose: prefetch
Accept-Encoding: gzip, deflate
Accept-Language: vi-VN,vi;q=0.9,fr-FR;q=0.8,fr;q=0.7,en-US;q=0.6,en;q=0.5
```

Figure 2.1: Một request cơ bản điển hình

Những phần khác là những chi tiết mà để xử lý chúng đòi hỏi người cài đặt phải có kỹ thuật cao. Trong khuôn khổ đồ án này ta sẽ không bàn tới.

2.2 Loại kết nối

[1] Khi tương tác server-client diễn ra trên giao thức TCP, người phát triển ứng dụng cần đưa ra một quyết định quan trọng: mỗi cặp request/response (yêu cầu/ phản hồi) sẽ sử dụng một kết nối TCP **chung** hay mỗi cặp cặp request/response sẽ sử dụng một kết nối TCP **riêng**. Theo cách ban đầu, ứng dụng được cho là sử dụng kết nối thường trực [persistent connections], còn theo cách sau thì gọi là kết nối không thường trực [non-persisitent connection].

Do kết nối thường trực có cách cài đặt tương đối khó khăn, chương trình trong đồ án này sẽ lựa chọn non-persistent connections. Vì vậy, mỗi đối tượng sẽ được gửi trong một phiên giao tiếp TCP giữa client-server, sau khi gửi xong thì kết nối TCP này cũng tự động ngắt kết nối.

Theo đó, mỗi một object trong một file html (bao gồm cả file html) sẽ được chuyển từ server đến client trong từng phiên riêng. Sau khi kết thúc việc truyền 1 object, server sẽ đóng socket truyền object đó bằng lệnh close().

2.3 CTE

Đồ án lựa chọn một cách thức encode đơn giản nhưng hiệu quả đó chính là chunked transfer encoding (gọi tắt là CTE). Theo đó một thông điệp chứa dữ liệu mà client yêu cầu sẽ được cắt nhỏ ra từng chunk (từng khối) rồi gửi từng phần đi thay vì gửi hết tất cả dữ liệu trong một lần gửi.

2.4 Cấu trúc response

2.4.1 Method của request là GET

Dựa vào kiến thức về CTE ở trên, ta tiến hành xây dựng cấu trúc response.
HTTP/1.1 200 OK
Content-Type: <tên kiểu file chuyền>
Transfer-encoding:Chunked-encoding
<kích thước chunk01 viết dưới dạng hex> $|r|n$
<data của chunk01>$|r|n$
<kích thước chunk02 viết dưới dạng hex> $|r|n$
<data của chunk02>$|r|n$

. . .
0 $|r|n$ (chunk gửi cuối cùng luôn là chunk rỗng nên kích thước là 0)
<data của chunk cuối >$|r|n$(rỗng)
$|r|n$(dấu hiệu kết thúc việc gửi chunk, lúc này client sẽ tự đóng socket)

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

7\r\n
Mozilla\r\n
9\r\n
Developer\r\n
7\r\n
Network\r\n
0\r\n
\r\n
```

Figure 2.2: Ví dụ cho response chuyền dữ liệu theo phương thức CTE[3]

Chi tiết cách cài đặt : Lần send đầu tiên, server sẽ tiến hành send header. Sau đó, server sẽ thực hiện lặp: lần lượt đọc dữ liệu file cần gửi, send kích thước chunk, rồi send data của chunk cho đến khi dữ liệu của file được đọc hết. Sau đó send chuỗi \r\n để client biết kết thúc chunk và đóng socket.

Lưu ý: để xử lý trường hợp client yêu cầu file không có thực, ta sẽ kiểm tra liệu file có tồn tại không bằng cách mở và đóng file. Nếu xảy ra lỗi, ta sẽ gửi lại 404 Not found response (sẽ được trình bày ở phần sau).

```
    if(method == "GET"):
        file_name=request_lines[0].split(' ')[1]
        file_name = file_name[1:]
        if(file_name==""):
            file_name="index.html"
        # file=open(file_name,"rb")
        # response_data=file.read()
        response_head="HTTP/1.1 200 OK\n"
    elif(method == "POST"):      You, a day ago • socket done
    if(file_name.endswith(".html")):
        response_head+="Content-Type: text/html\n"
    elif(file_name.endswith(".png")):
        response_head+="Content-Type: image/png\n"
    response_head +="Transfer-Encoding: chunked\n\n"
    print(response_head)
    print(response_head.encode())
    try:
        f=open(file_name,"rb")
        f.close()
    except Exception as e:
        send404(stream)
        continue
    stream.send(response_head.encode())
    sendFile_CTE(file_name,stream)
    stream.close()
```

Figure 2.3: Gửi phần header cho client trước khi gửi phần data

```python
def sendFile_CTE(file_name,stream):
    file=open(file_name,"rb")
    while True:
        chunk = file.read(65536)
        stream.sendall(bytes("%s\r\n" % hex(len(chunk))[2:],"utf8"))
        #print(bytes("%s\r\n" % hex(len(chunk))[2:],"utf8"))#debug
        chunk+=bytes("\r\n","utf8")
        stream.sendall(chunk)
        #print(chunk)#debug
        if len(chunk) == 2: #string only has 2 char "\r\n", that is empty string : break out
            break
        #time.sleep(1)#debug

    #CTE flow ends with 2 character "\r\n"
    stream.send(bytes("\r\n","utf8"))
```

Figure 2.4: Cài đặt hàm chuyển data từ file theo phương thức CTE bằng ngôn ngữ python

2.4.2 Method của request là POST

Trong đồ án này, giao thức POST được sử dung trong việc nhập username và password. Khi đó, ta sẽ kiểm tra username và password, nếu đúng thì trả về response với mã 301 để bên client gửi lại request file biểu diễn thông tin theo yêu cầu đề bài, lúc này lại trở về việc giải quyết method GET như trên. Trượng hợp nhập username và password sai, ta sẽ gửi thông điệp 404 sẽ được trình bày ở phần tiếp theo.

```
elif(method == "POST"):      You, a day ago • socket done
    message = request_lines[-1]
    print(message)
    if(isCorrestLogin(message)):
        header = '''HTTP/1.1 301 Moved Permanently\nLocation:/post.html'''
        stream.send(header.encode())
        stream.close()
        continue
    else:
        send404(stream)
    continue
```

Figure 2.5: Cài đặt xử lý method POST

2.4.3 404 - Notfound Response

Ngoài việc thay đổi dòng header và mặc định gửi dữ liệu file html trình chiếu trang 404, 404 response không có sự thay đổi quá nhiều so với response GET.

```
def send404(stream):
    header = '''HTTP/1.1 404 Not Found\nContent-Type: text/html\n\n'''
    file_name="404.html"
    file=open(file_name,"rb")
    response_data=file.read()
    stream.sendall(header.encode()+response_data)
    stream.close()
```

Figure 2.6: Cài đặt gửi 404 response

Bibliography

[1] James F. Kurose. *Computer Networking: A Top-Down Approach.* 2016.

[2] David .*Python Network Programming.* Thu Jun 17 19:49:58 2010.

[3] *https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding.*

[4] Trần Đan Thư, Nguyễn Thanh Phương, Đinh Bá Tiến, Trần Minh Triết, Đặng Bình Phương. *Kỹ thuật lập trình,Chương 8 : Lập trình đệ quy.* Khoa Công nghệ thông tin, Trường ĐHKHTN TPHCM, 2014.