TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

—————o0o—————

BÁO CÁO MÔN HỌC
MẠNG MÁY TÍNH

PROXY

Thành viên	MSSV
Đặng Minh Ánh	23120418
Trần Thế Bảo	23120002
Đặng Thanh Long	23120142

Ngày 17 tháng 12 năm 2024

Bảng phân công công việc

Họ và tên	Mã số sinh viên	Tỷ lệ đóng góp
Đặng Minh Ánh	23120418	100%
Đặng Thanh Long	23120142	100%
Trần Thế Bảo	23120002	100%

Mục lục

1 Khởi tạo

1.1 Thư viện:

```
#include <iostream>
#include <winsock2.h>
#pragma comment(lib, "ws2_32.lib")

#define PORT 8080           // Port for the proxy server
#define BUFFER_SIZE 10000
```

1.2 Khởi tạo Winsock

```
WSADATA wsadata;

if (WSAStartup(MAKEWORD(2, 2), &wsadata) != 0) {
    std::cerr << "Winsock initialization failed!\n";
    return 0;
}

cout << "Winsock initialized successfully.\n";
```

1.3 Khởi tạo Proxy Server

```
SOCKET proxyServer = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (proxyServer == INVALID_SOCKET) {
    cerr << "Failed to create server socket, error: " << WSAGetLastError()
       << '\n';
    WSACleanup();
    return -1;
}

sockaddr_in proxyAddress{};
proxyAddress.sin_family = AF_INET;
InetPton(AF_INET, "127.0.0.1", &proxyAddress.sin_addr.s_addr);
proxyAddress.sin_port = htons(PORT);
```

- **Tạo socket:**
  - Dùng hàm `socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)` để tạo một socket TCP sử dụng IPv4.
  - Kiểm tra lỗi: Nếu socket không hợp lệ (`INVALID_SOCKET`), in lỗi và thoát chương trình.
- **Cấu hình địa chỉ:**

- `sin_family = AF_INET`: Thiết lập kiểu địa chỉ IPv4.
- `InetPton(AF_INET, "127.0.0.1", &proxyAddress.sin_addr.s_addr)`: Chuyển chuỗi địa chỉ IP "127.0.0.1" thành dạng nhị phân.
- `sin_port = htons(PORT)`: Chuyển đổi cổng từ dạng host byte order sang network byte order.

Ý chính: Đoạn mã khởi tạo một socket TCP và cấu hình địa chỉ `localhost` với cổng xác định (`PORT`) để chuẩn bị cho server hoặc proxy hoạt động.

2 Kết nối

2.1 Bind

```
if (bind(proxyServer, (SOCKADDR*)&proxyAddress, sizeof(proxyAddress)) ==
    SOCKET_ERROR) {
    cerr << "Failed to bind socket" << '\n';
    closesocket(proxyServer);
    WSACleanup();
    return -1;
}
```

- `bind()` là hàm được sử dụng để kết nối một socket với một địa chỉ cụ thể (được lưu trong `proxyAddress`).
- Nếu `bind()` trả về `SOCKET_ERROR`, điều đó có nghĩa là việc kết nối không thành công.
- Trong trường hợp lỗi, thông báo `"Failed to bind socket"` được in ra màn hình, và socket sẽ bị đóng với `closesocket()`. Cuối cùng, `WSACleanup()` được gọi để giải phóng tài nguyên Winsock, và chương trình sẽ kết thúc với mã lỗi `-1`.

2.2 Listen

```
if (listen(proxyServer, SOMAXCONN) == SOCKET_ERROR) {
    cerr << "Failed to listen on socket" << '\n';
    closesocket(proxyServer);
    WSACleanup();
    return -1;
}
```

- `listen()` là hàm được sử dụng để chuẩn bị socket cho việc chấp nhận các kết nối từ các client. Tham số thứ hai `SOMAXCONN` chỉ định số lượng kết nối tối đa có thể chờ đợi trong hàng đợi.
- Nếu `listen()` trả về `SOCKET_ERROR`, điều đó có nghĩa là việc "lắng nghe"trên socket thất bại.
- Trong trường hợp lỗi, thông báo `"Failed to listen on socket"` được in ra màn hình, và socket sẽ bị đóng với `closesocket()`. Cuối cùng, `WSACleanup()` được gọi để giải phóng tài nguyên Winsock, và chương trình sẽ kết thúc với mã lỗi `-1`.

3 Xử lí request

3.1 HTTP/HTTPS request

Một HTTPS request gồm 4 phần chính:

1. **Request Line**: Chứa phương thức, đường dẫn, và phiên bản HTTP.
   Ví dụ: `GET /index.html HTTP/1.1`

2. **Request Headers**: Cung cấp thông tin bổ sung dưới dạng `key: value`.
   Ví dụ:

```
Host: www.example.com
User-Agent: Mozilla/5.0
```

3. **Empty Line**: Phân tách phần headers và body.

4. **Request Body**: Chứa dữ liệu gửi lên server (cho `POST`, `PUT`).

Ví dụ:

```
POST /api/login HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Content-Type: application/json
Content-Length: 42

{
    "username": "john_doe",
    "password": "123456"
}
```

3.2 Remote Server (Máy chủ từ xa)

Khi một người dùng muốn truy cập một website hoặc dịch vụ, proxy server sẽ tiếp nhận yêu cầu của người dùng, thay vì gửi trực tiếp yêu cầu đó đến máy chủ của website. Proxy server sẽ gửi yêu cầu đến máy chủ từ xa (remote server), nhận phản hồi từ máy chủ đó, và sau đó chuyển tiếp dữ liệu về cho người dùng. Do đó, remote server (máy chủ thực tế xử lý các yêu cầu) là yếu tố quan trọng trong quá trình này.

3.2.1 Địa chỉ IP của Host

Chuyển host từ domain name sang IP là một bước quan trọng trong quá trình truy cập một website hoặc dịch vụ trên Internet. Request mà ta nhận được từ HTTP/HTTPS request sẽ gửi

trang web dưới định dạng domain name, ta cần chuyển sang IP để máy tính có thể giao tiếp với máy chủ.

```
struct hostent* he = gethostbyname(host.c_str());
if (he == nullptr) {
    cerr << "Host resolution failed!\n";
    closesocket(remoteSocket);
    return;
}
```

3.2.2 Kết nối với Host

Đoạn mã dưới đây thiết lập kết nối đến một máy chủ từ xa thông qua một socket TCP:

```
sockaddr_in server_addr{};
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
memcpy(&server_addr.sin_addr, he->h_addr_list[0], he->h_length);

// Connect to server
if (connect(remoteSocket, (struct sockaddr*)&server_addr,
   sizeof(server_addr)) == SOCKET_ERROR) {
    cerr << "Failed to connect to server" << '\n';
    closesocket(remoteSocket);
    return;
}
```

Giải thích:

- `sockaddr_in server_addr{};`: Khai báo và khởi tạo cấu trúc chứa thông tin về địa chỉ của máy chủ.
- `server_addr.sin_family = AF_INET;`: Thiết lập giao thức IPv4.
- `server_addr.sin_port = htons(port);`: Thiết lập cổng máy chủ, chuyển đổi sang định dạng mạng.
- `memcpy(server_addr.sin_addr, he->h_addr_list[0], he->h_length);`: Sao chép địa chỉ IP của máy chủ từ thông tin DNS.
- `connect(remoteSocket, (struct sockaddr*)server_addr, sizeof(server_addr))`: Kết nối socket đến máy chủ với thông tin đã cấu hình.
- Nếu kết nối thất bại (trả về `SOCKET_ERROR`), thông báo lỗi được in ra và socket bị đóng.

3.2.3 Response

```
// send respone to browser that connection had established
string connect_response = "HTTP/1.1 200 Connection Established\r\n\r\n";
int bytesSent = send(clientSocket, connect_response.c_str(),
   connect_response.size(), 0);
```

- HTTP/1.1 200: Mã trạng thái 200 cho biết rằng yêu cầu đã thành công.
- Connection Established: Thông báo rằng kết nối giữa client và server từ xa đã được thiết lập thành công, và proxy server không can thiệp vào dữ liệu truyền tải giữa chúng

Đoạn mã trên được sử dụng trong một proxy server, và mục đích của nó là gửi phản hồi từ máy chủ (proxy) tới client (browser) để thông báo rằng kết nối đã được thiết lập thành công.
Tại sao cần phản hồi này?

1. Khi trình duyệt gửi một yêu cầu HTTP CONNECT đến proxy (thường xảy ra trong các kết nối HTTPS), proxy cần phải thông báo cho trình duyệt rằng nó đã thiết lập được kết nối đến server mà client yêu cầu. Mã 200 OK này là cách thông báo cho trình duyệt biết rằng kết nối đã được thiết lập thành công, và bây giờ có thể tiến hành truyền tải dữ liệu mã hóa (trong trường hợp của HTTPS).
2. Nếu không gửi phản hồi này, trình duyệt sẽ không biết rằng proxy đã kết nối thành công tới server từ xa và sẽ không tiếp tục giao tiếp với máy chủ.

4 Truyền tải dữ liệu

Sau khi proxy server gửi phản hồi HTTP/1.1 200 Connection Established cho client (trình duyệt), kết nối giữa client và remote server đã được thiết lập. Từ thời điểm này, quá trình truyền tải dữ liệu giữa client và remote server sẽ diễn ra trực tiếp qua proxy mà không có sự can thiệp của proxy trong việc xử lý dữ liệu. Tuy nhiên, proxy vẫn đóng vai trò là kênh trung gian cho dữ liệu truyền đi.

Các bước chi tiết trong quá trình truyền tải dữ liệu sau khi "Connection Established":

1. Client gửi dữ liệu đến Proxy:
   - Sau khi nhận được phản hồi "Connection Established", client (trình duyệt) sẽ bắt đầu gửi dữ liệu đến proxy server
   - Trong trường hợp của HTTPS, dữ liệu này là mã hóa và sẽ không thể đọc được bởi proxy.
2. Proxy chuyển tiếp dữ liệu đến Remote Server:
   - Proxy server, sau khi nhận dữ liệu từ client qua socket clientSocket, sẽ chuyển tiếp (forward) dữ liệu này tới remote server (máy chủ đích) qua socket remoteSocket.
   - Proxy sẽ không giải mã hoặc chỉnh sửa dữ liệu, mà chỉ chuyển tiếp byte dữ liệu từ client đến server và ngược lại.
3. Remote Server gửi dữ liệu phản hồi đến Proxy:
   - Khi remote server (máy chủ đích) nhận được dữ liệu từ proxy, nó sẽ xử lý và trả về dữ liệu phản hồi (ví dụ như dữ liệu web, hình ảnh, hoặc các tài nguyên khác).
4. Proxy chuyển tiếp dữ liệu từ Remote Server đến Client:
   - Proxy server nhận dữ liệu phản hồi từ remote server qua socket remoteSocket và gửi lại dữ liệu này tới client qua socket clientSocket.
   - Quá trình này diễn ra liên tục và proxy không thay đổi nội dung, chỉ đơn giản là truyền tải dữ liệu giữa hai bên.
5. Kết thúc kết nối:
   - Khi một trong hai bên (client hoặc remote server) kết thúc kết nối (bằng cách đóng socket hoặc kết thúc phiên giao dịch), proxy server cũng cần phải đóng các kết nối tương ứng.

- Trong trường hợp client đóng kết nối hoặc kết nối bị lỗi, proxy sẽ dừng việc chuyển tiếp dữ liệu và đóng các socket liên quan.

Trong trường hợp client đóng kết nối hoặc kết nối bị lỗi, proxy sẽ dừng việc chuyển tiếp dữ liệu và đóng các socket liên quan.

```
// Step 1: Client send data to proxy
if (FD_ISSET(clientSocket, &readfds)) {
    bytes_read = recv(clientSocket, buffer, BUFFER_SIZE, 0); // Receive
       data from client
    if (bytes_read <= 0) {
        break; // Client disconnected
    }

    // Step 2: Proxy send data to remote server
    bytes_sent = send(remoteSocket, buffer, bytes_read, 0); // Send data to
       server
    if (bytes_sent <= 0) {
        break; // Error sending data
    }
}

// Step 3: Server response data to proxy
if (FD_ISSET(remoteSocket, &readfds)) {
    bytes_read = recv(remoteSocket, buffer, BUFFER_SIZE, 0); // Receive
       data from server
    if (bytes_read <= 0) {
        break; // Server disconnected
    }

    // Step 4: Proxy send response fron server to client
    bytes_sent = send(clientSocket, buffer, bytes_read, 0); // Send data
       from server to client
    if (bytes_sent <= 0) {
        break; // Error sending data
    }
}
```