LẬP TRÌNH MẠNG
(IT4060)
Network Programming

Giảng viên: Lê Bá Vui
Email: vuilb@soict.hust.edu.vn; vui.leba@hust.edu.vn
Bộ môn KTMT – Viện CNTT & TT

Mục đích môn học

- Cung cấp kiến thức cơ bản về lập trình ứng dụng trên mạng:
  – Xây dựng ứng dụng phía server.
  – Xây dựng ứng dụng phía client.
  – Các kỹ thuật vào ra.
  – Tìm hiểu và thực hiện một số giao thức chuẩn.
- Cung cấp các kỹ năng cần thiết để thiết kế và xây dựng ứng dụng mạng:
  – Sử dụng thư viện, môi trường, tài liệu.
  – Thiết kế, xây dựng chương trình.

Yêu cầu

- Yêu cầu về kiến thức nền tảng:
  - Mạng máy tính: địa chỉ IP, tên miền, giao thức, …
  - Ngôn ngữ lập trình: **C**, **C++**
  - Các kỹ thuật lập trình: mảng, chuỗi ký tự, con trỏ, cấp phát bộ nhớ động, …
  - Các kỹ năng lập trình, gỡ lỗi
- Yêu cầu khác:
  - Lên lớp đầy đủ
  - Hoàn thành bài tập về nhà
  - *Hoàn thành bài tập lớn*
- **Điểm quá trình** = Điểm thi giữa kỳ + Điểm danh + Bài tập về nhà + *Bài tập lớn*
- **Điểm cuối kỳ** = Điểm thi cuối kỳ

Tài liệu tham khảo

- **Slide bài giảng**
- **Network Programming for Microsoft Windows Second Edition.** ***Anthony Jone, Jim Ohlun.***
- Google, StackOverflow, …

Link tải bài giảng và các ví dụ:
http://bit.ly/IT4060

Nội dung môn học

Chương 1. Giới thiệu về lập trình mạng
Chương 2. Lập trình socket
Chương 3. Giới thiệu về lập trình đa luồng
Chương 4. Các phương pháp vào ra trong lập trình socket
Chương 5. Tìm hiểu và cài đặt một số giao thức phổ biến

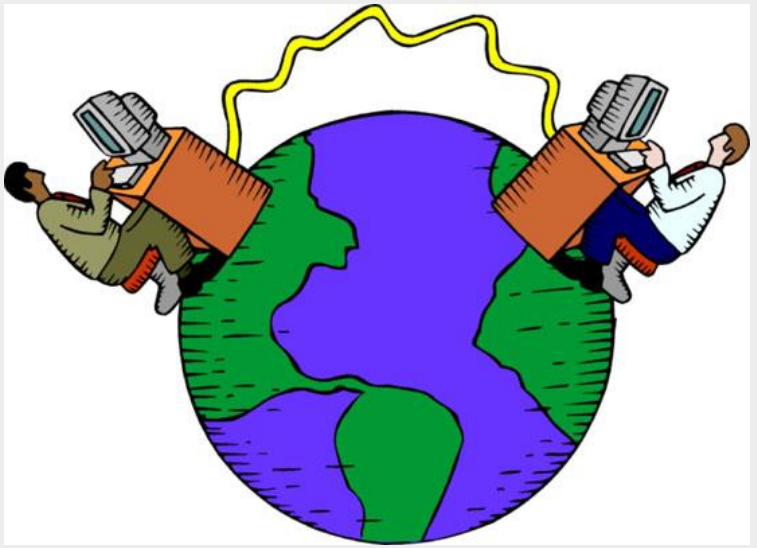
Chương 1. Giới thiệu về Lập trình mạng

Chương 1. Giới thiệu về lập trình mạng

1.1. Khái niệm
1.2. Giao thức Internet
1.3. Giao thức TCP
1.4. Giao thức UDP
1.5. Hệ thống phân giải tên miền

1.1. Khái niệm

Lập trình mạng bao gồm các kỹ thuật lập trình nhằm xây dựng ứng dụng, phần mềm với mục đích khai thác hiệu quả tài nguyên mạng máy tính.



1.1. Khái niệm

- Các vấn đề cần phải quan tâm:
  - Thông tin truyền nhận trên mạng
  - Các giao thức truyền thông (Protocols)
    - Giao thức chuẩn (HTTP, FTP, SMTP, …)
    - Giao thức tự định nghĩa
  - **Các kỹ thuật truyền nhận dữ liệu**
  - Các kỹ thuật nâng cao:
    - Nén dữ liệu
    - Mã hóa dữ liệu
    - Truyền nhận dữ liệu song song

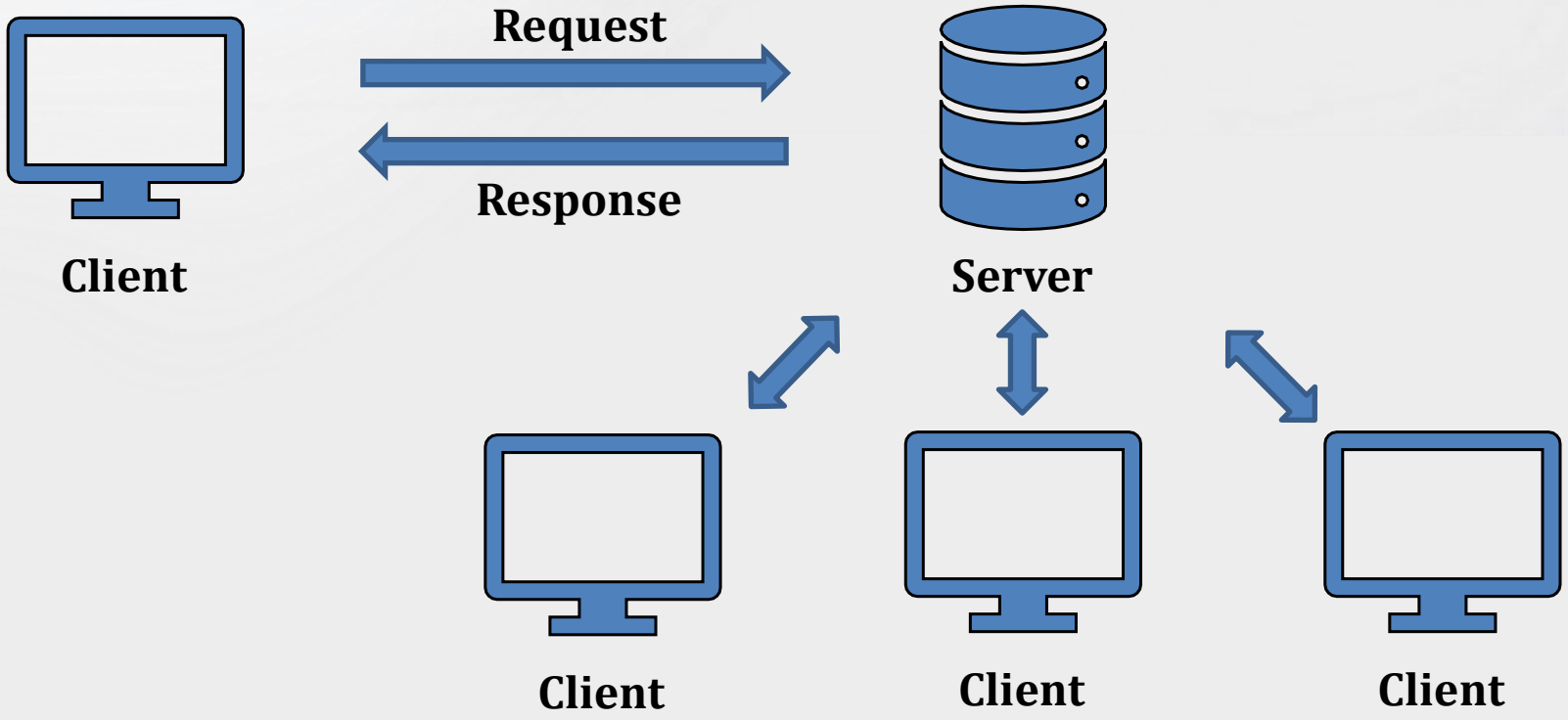
1.1. Khái niệm

- Các ngôn ngữ được sử dụng để lập trình mạng
  - **C/C++**: Mạnh và phổ biến, được hầu hết các lập trình viên sử dụng để viết các ứng dụng mạng hiệu năng cao.
  - **Java**: Khá thông dụng, sử dụng nhiều trong các điện thoại di động (J2ME, Android).
  - **C#**: Mạnh và dễ sử dụng, tuy nhiên chạy trên nền .Net Framework và chỉ hỗ trợ họ hệ điều hành Windows.
  - **Python, Perl, PHP**... Ngôn ngữ thông dịch, sử dụng để viết các tiện ích nhỏ, nhanh chóng.

⇒ Giáo trình này sẽ chỉ đề cập đến hai ngôn ngữ **C/C++**

1.1. Khái niệm

- Mô hình server – client



1.1. Khái niệm

- Các kiểu ứng dụng hoạt động trên mạng
  - Các ứng dụng máy chủ (servers)
    - HTTP, FTP, Mail server
    - Game server
    - Media server (DLNA), Streaming server (video, audio)
    - Proxy server
  - Các ứng dụng máy khách (clients)
    - Game online
    - Mail client, FTP client, Web client
  - Các ứng dụng mạng ngang hàng
    - uTorrent
  - Các ứng dụng khác
    - Internet Download Manager
    - Microsoft Network Monitor, WireShark
    - Firewall

1.1. Khái niệm

- Ví dụ về các ứng dụng trên mạng
  - Phần mềm web
    - Client (browser) gửi các yêu cầu đến web server
    - Web server thực hiện yêu cầu và trả lại kết quả cho trình duyệt
  - Phần mềm chat
    - Server quản lý dữ liệu người dùng
    - Client gửi các yêu cầu đến server (đăng ký, đăng nhập, các đoạn chat)
    - Server thực hiện yêu cầu và trả lại kết quả cho client
      - Đăng ký?
      - Đăng nhập
      - Chuyển tiếp dữ liệu giữa các client

1.1. Khái niệm

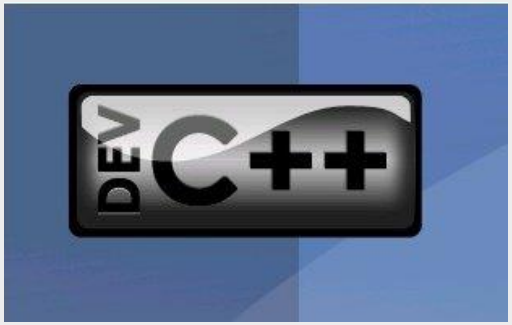
- Ví dụ về các ứng dụng trên mạng
  - Phần mềm nghe nhạc trên thiết bị di động (Spotify)
    - Server quản lý dữ liệu người dùng, lưu trữ các file âm thanh, xử lý các yêu cầu từ phần mềm di động, quản lý các kết nối.
    - Phần mềm di động gửi các yêu cầu và dữ liệu lên server, chờ kết quả trả về và xử lý.
  - Phần mềm đồng bộ file giữa các thiết bị (Dropbox, Onedrive, …)
    - Cài đặt phần mềm client trên PC
    - Đồng bộ thư mục và tập tin lên server
    - Theo dõi sự thay đổi của dữ liệu (từ phía server hoặc local) và cập nhật theo thời gian thực
  - Phần mềm tăng tốc download IDM
    - Bắt và phân tích các gói tin được nhận bởi trình duyệt
    - Tách ra các liên kết quan tâm
    - Tải file bằng nhiều luồng song song

1.1. Khái niệm

- Thư viện được sử dụng:
  - **Windows Socket API** (WinSock)
    - Thư viện liên kết động (WS2_32.DLL) đi kèm trong hệ điều hành Windows của Microsoft
    - Thường sử dụng cùng với C/C++
    - Cho hiệu năng cao nhất
  - **MFC Socket**
    - Viết lại thư viện WinSock dưới dạng các lớp hướng đối tượng
  - **System.Net** và **System.Net.Sockets**
    - Hai namespace trong bộ thư viện .NET của Microsoft
    - Dễ sử dụng
    - Thường sử dụng với C#

1.1. Khái niệm

- Các công cụ lập trình
  - **Visual Studio (2019, 2017, ...)**
    - Rất mạnh
    - Tích hợp nhiều công cụ lập trình
    - Hỗ trợ cả WinSock, MFC Socket và .NET Socket
    - Bản Community được tải miễn phí
  - **Dev C++**
    - Miễn phí
    - Chỉ hỗ trợ WinSock



1.1. Khái niệm

- Công cụ gỡ lỗi
  - TCPView: Hiển thị các kết nối hiện tại của máy tính.
  - Resource Monitor: ~ TCPView.
  - **Wireshark**, Microsoft Network Monitor
  - **Netcat** (Netcat Win32)

1.1. Khái niệm

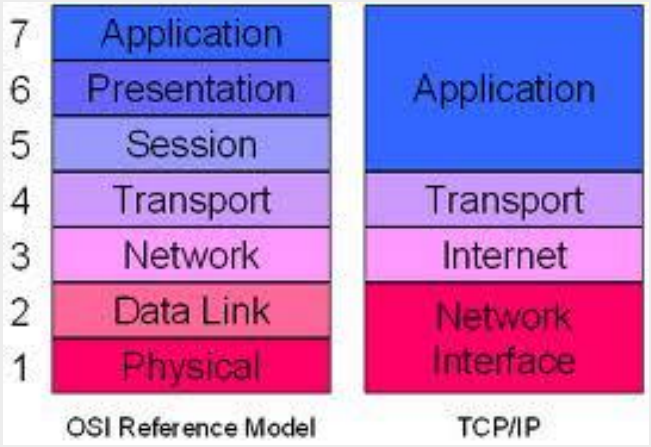
- Tài liệu tra cứu
  - **Microsoft Developer Network – MSDN**
    - Cực kỳ chi tiết và chuyên nghiệp
    - Công cụ không thể thiếu
  - **Google/BING**
  - **Stack Overflow**

1.2. Bộ giao thức Internet (IP)

a. Giới thiệu
b. Giao thức IPv4
c. Giao thức IPv6

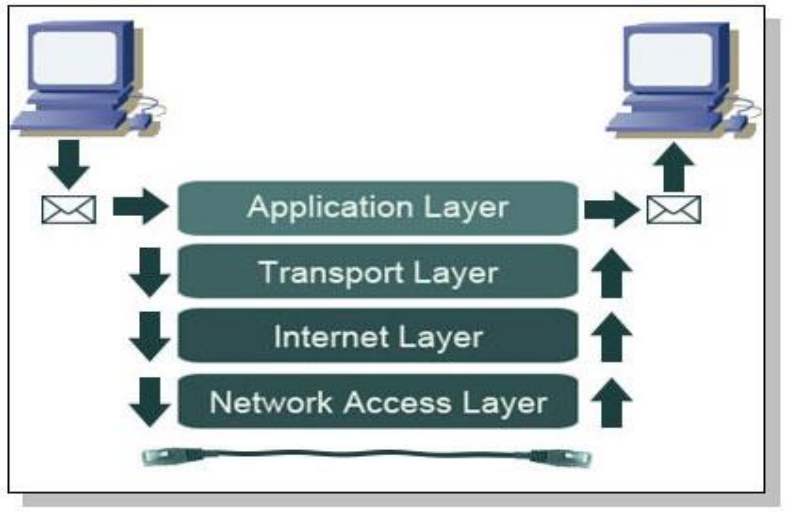
a. Giới thiệu

- Bộ giao thức Internet
  - TCP/IP: Transmission Control Protocol/Internet Protocol.
  - Là bộ giao thức truyền thông được sử dụng trên Internet và hầu hết các mạng thương mại.
  - Được chia thành các tầng gồm nhiều giao thức, thuận tiện cho việc quản lý và phát triển.
  - Là thể hiện đơn giản hóa của mô hình lý thuyết OSI.



a. Giới thiệu

- Bộ giao thức Internet
  - Gồm bốn tầng
    - Tầng ứng dụng – Application Layer.
    - Tầng giao vận – Transport Layer.
    - Tầng Internet – Internet Layer.
    - Tầng truy nhập mạng – Network Access Layer.



a. Giới thiệu

- Bộ giao thức Internet
  - Tầng ứng dụng
    - Đóng gói dữ liệu người dùng theo giao thức riêng và chuyển xuống tầng dưới.
    - Các giao thức thông dụng: HTTP, FTP, SMTP, POP3, DNS, SSH, IMAP...
    - *Việc lập trình mạng sẽ xây dựng ứng dụng tuân theo một trong các giao thức ở tầng này hoặc giao thức do người phát triển tự định nghĩa*

a. Giới thiệu

- Bộ giao thức Internet
  - Tầng giao vận
    - Cung cấp dịch vụ truyền dữ liệu giữa ứng dụng - ứng dụng.
    - Đơn vị dữ liệu là các đoạn (segment, datagram)
    - Các giao thức ở tầng này: TCP, UDP.
    - *Việc lập trình mạng sẽ sử dụng dịch vụ do các giao thức ở tầng này cung cấp để truyền dữ liệu*

a. Giới thiệu

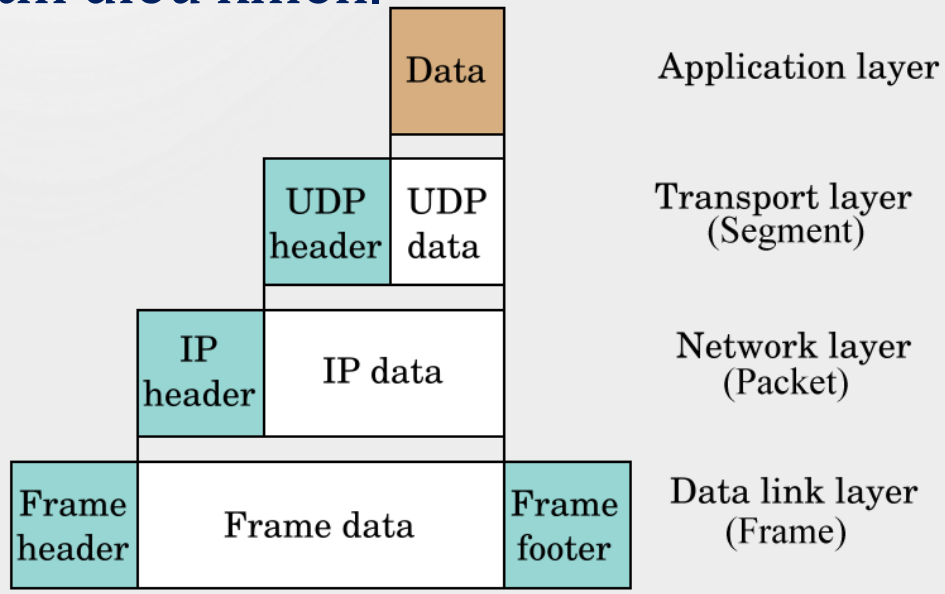
- Bộ giao thức Internet
  - Tầng Internet
    - Định tuyến và truyền các gói tin liên mạng.
    - Cung cấp dịch vụ truyền dữ liệu giữa máy tính – máy tính trong cùng nhánh mạng hoặc giữa các nhánh mạng.
    - Đơn vị dữ liệu là các gói tin (packet).
    - Các giao thức ở tầng này: IPv4, IPv6
    - *Việc lập trình ứng dụng mạng sẽ rất ít khi can thiệp vào tầng này, trừ khi phát triển một giao thức liên mạng mới.*

a. Giới thiệu

- Bộ giao thức Internet
  - Tầng truy nhập mạng
    - Cung cấp dịch vụ truyền dữ liệu giữa các nút mạng trên cùng một nhánh mạng vật lý.
    - Đơn vị dữ liệu là các khung (frame).
    - Phụ thuộc rất nhiều vào phương tiện kết nối vật lý.
    - Các giao thức ở tầng này đa dạng: MAC, LLC, ADSL, 802.11...
    - *Việc lập trình mạng ở tầng này là xây dựng các trình điều khiển phần cứng tương ứng, thường do nhà sản xuất thực hiện.*

a. Giới thiệu

- Bộ giao thức Internet
  - Dữ liệu gửi đi qua mỗi tầng sẽ được thêm phần thông tin điều khiển (header).
  - Dữ liệu nhận được qua mỗi tầng sẽ được bóc tách thông tin điều khiển.



a. Giới thiệu

- Giao thức Internet (Internet Protocol)
  - Giao thức mạng thông dụng nhất trên thế giới
  - Chức năng
    - Định địa chỉ các máy chủ
    - Định tuyến các gói dữ liệu trên mạng
  - Bao gồm 2 phiên bản: IPv4 và IPv6
  - Thành công của Internet là nhờ IPv4
  - Được hỗ trợ trên tất cả các hệ điều hành
  - Là công cụ sử dụng để lập trình ứng dụng mạng

b. Giao thức IPv4

- Giao thức IPv4
  - Được IETF công bố dưới dạng RFC 791 vào 9/1981.
  - Phiên bản thứ 4 của họ giao thức IP và là phiên bản đầu tiên phát hành rộng rãi.
  - Sử dụng trong hệ thống **chuyển mạch gói**.
  - Truyền dữ liệu theo kiểu **Best-Effort:** không đảm bảo tính trật tự, trùng lặp, tin cậy của gói tin.
  - Kiểm tra tính toàn vẹn của dữ liệu qua **checksum**

b. Giao thức IPv4

- IPv4 header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

Một số trường cần quan tâm:
Version (4 bit): có giá trị là 4 với IPv4
IHL – Internet Header Length (4 bit): chiều dài của header, tính bằng số từ nhớ 32 bit
Total Length (16 bit): kích thước của gói tin (theo bytes) bao gồm cả header và data
Protocol: giao thức được sử dụng ở tầng trên (nằm trong phần data)
Source IP Address: địa chỉ IP nguồn
Destination IP Address: địa chỉ IP đích

b. Giao thức IPv4

- IPv4 header – Ví dụ

Gói tin IPv4 có header như sau:

```
45 00 00 40
7c da 40 00
80 06 fa d8
c0 a8 0f 0b
bc ac f6 a4
```

Xác định các thông tin liên quan đến gói tin này:
Header length, Total length, Protocol, Source IP, Destination IP address.

b. Giao thức IPv4

- Địa chỉ IPv4
  - Sử dụng 32 bit để đánh địa chỉ các máy tính trong mạng.
  - Bao gồm: phần mạng và phần host.
  - Số địa chỉ tối đa: $2^{32} \sim 4{,}294{,}967{,}296$.
  - Dành riêng một vài dải đặc biệt không sử dụng.
  - Chia thành bốn nhóm 8 bít (octet).

Dạng biểu diễn	Giá trị
Nhị phân	11000000.10101000.00000000.00000001
Thập phân	192.168.0.1
Thập lục phân	0xC0A80001

b. Giao thức IPv4

- Các lớp địa chỉ IPv4
  - Có năm lớp địa chỉ: A, B, C, D, E.
  - Lớp A, B, C: trao đổi thông tin thông thường.
  - Lớp D: **multicast**
  - Lớp E: để dành

Lớp	MSB	Địa chỉ đầu	Địa chỉ cuối
A	0xxx	0.0.0.0	127.255.255.255
B	10xx	128.0.0.0	191.255.255.255
C	110x	192.0.0.0	223.255.255.255
D	1110	224.0.0.0	239.255.255.255
E	1111	240.0.0.0	255.255.255.255

b. Giao thức IPv4

- Mặt nạ mạng (Network Mask)
  - Phân tách phần mạng và phần host trong địa chỉ IPv4.
  - Sử dụng trong bộ định tuyến để tìm đường đi cho gói tin.
  - Với mạng có dạng

Network	Host
192.168.0.	1
11000000.10101000.00000000.	00000001

b. Giao thức IPv4

- Mặt nạ mạng (Network Mask)
  - Biểu diễn theo dạng **/n**
    - **n** là số bit dành cho phần mạng.
    - Thí dụ: 192.168.0.1/24
  - Biểu diễn dưới dạng nhị phân
    - Dùng 32 bit đánh dấu, bít dành cho phần mạng là 1, cho phần host là 0.
    - Thí dụ: 11111111.11111111.11111111.00000000 hay 255.255.255.0
  - Biểu diễn dưới dạng Hexa
    - Dùng số Hexa: 0xFFFFFF00
    - Ít dùng

b. Giao thức IPv4

- Số lượng địa chỉ trong mỗi mạng
  - Mỗi mạng sẽ có n bit dành cho phần mạng, 32-n bit dành cho phần host.
  - Phân phối địa chỉ trong mỗi mạng:
    - 01 địa chỉ mạng (các bit phần host bằng 0).
    - 01 địa chỉ quảng bá (các bit phần host bằng 1).
    - 2^n-2 địa chỉ gán cho các máy trạm (host).
  - Với mạng 192.168.0.1/24
    - Địa chỉ mạng: 192.168.0.0
    - Địa chỉ quảng bá: 192.168.0.255
    - Địa chỉ host: 192.168.0.1 - 192.168.0.254

b. Giao thức IPv4

- Các dải địa chỉ đặc biệt
  - Là những dải được dùng với mục đích riêng, không sử dụng được trên Internet.

Địa chỉ	Diễn giải
10.0.0.0/8	Mạng riêng
127.0.0.0/8	Địa chỉ loopback
172.16.0.0/12	Mạng riêng
192.168.0.0/16	Mạng riêng
224.0.0.0/4	Multicast
240.0.0.0/4	Dự trữ

b. Giao thức IPv4

- Dải địa chỉ cục bộ
  - Chỉ sử dụng trong mạng nội bộ.
  - Khắc phục vấn đề thiếu địa chỉ của IPv4.

Tên	Dải địa chỉ	Số lượng	Mô tả mạng	Viết gọn
Khối 24-bit	10.0.0.0–10.255.255.255	16,777,216	Một dải trọn vẹn thuộc lớp A	10.0.0.0/8
Khối 20-bit	172.16.0.0–172.31.255.255	1,048,576	Tổ hợp từ mạng lớp B	172.16.0.0/12
Khối 16-bit	192.168.0.0–192.168.255.255	65,536	Tổ hợp từ mạng lớp C	192.168.0.0/16

c. Giao thức IPv6

- Giao thức IPv6
  - IETF đề xuất năm 1998.
  - Khắc phục vấn đề thiếu địa chỉ của IPv4.
  - Đang dần phổ biến và chưa thể thay thế hoàn toàn IPv4.
  - Sử dụng 128 bit để đánh địa chỉ các thiết bị, dưới dạng các cụm số hexa phân cách bởi dấu :

  Ví dụ: **FEDC:BA98:768A:0C98:FEBA:CB87:7678:1111**

c. Giao thức IPv6

- Quy tắc rút gọn địa chỉ IPv6
  - Cho phép bỏ các số 0 nằm trước mỗi nhóm (octet).
  - Thay bằng số 0 cho nhóm có toàn số 0.
  - Thay bằng dấu "::" cho các nhóm liên tiếp nhau có toàn số 0.

Chú ý: Dấu "::" chỉ sử dụng được 1 lần trong toàn bộ địa chỉ IPv6

Ví dụ: địa chỉ
1080:0000:0000:0070:0000:0989:CB45:345F
Có thể viết tắt thành **1080::70:0:989:CB45:345F** hoặc
1080:0:0:70::989: CB45:345F

1.3. Giao thức TCP

- Giao thức TCP: Transmission Control Protocol
  - Giao thức lõi chạy ở tầng giao vận.
  - Chạy bên dưới tầng ứng dụng và trên nền IP
  - Cung cấp dịch vụ truyền dữ liệu theo dòng tin cậy giữa các ứng dụng.
  - Được sử dụng bởi hầu hết các ứng dụng mạng.
  - Chia dữ liệu thành các gói nhỏ, thêm thông tin kiểm soát và gửi đi trên đường truyền.
  - *Lập trình mạng sẽ sử dụng giao thức này để trao đổi thông tin.*

1.3. Giao thức TCP

- Cổng (Port)
  - Một số nguyên duy nhất trong khoảng 0-65535 tương ứng với một kết nối của ứng dụng.
  - TCP sử dụng cổng để chuyển dữ liệu tới đúng ứng dụng hoặc dịch vụ.
  - Một ứng dụng có thể mở nhiều kết nối => có thể sử dụng nhiều cổng.
  - Một số cổng thông dụng: HTTP(80), FTP(21), SMTP(25), POP3(110), HTTPS(443)...

1.3. Giao thức TCP

- Đặc tính của TCP
  - Hướng kết nối: **connection oriented**
    - Hai bên phải thiết lập kênh truyền trước khi truyền dữ liệu.
    - Được thực hiện bởi quá trình gọi là bắt tay ba bước (three ways handshake).
  - Truyền dữ liệu theo dòng (**stream oriented**): tự động phân chia dòng dữ liệu thành các đoạn nhỏ để truyền đi, tự động ghép các đoạn nhỏ thành dòng dữ liệu và gửi trả ứng dụng.
  - Đúng trật tự (ordering guarantee): dữ liệu gửi trước sẽ được nhận trước

1.3. Giao thức TCP

- Đặc tính của TCP
  - Tin cậy, chính xác: thông tin gửi đi sẽ được đảm bảo đến đích, không dư thừa, sai sót...
  - Độ trễ lớn, khó đáp ứng được tính thời gian thực.

1.3. Giao thức TCP

- Header của TCP
  - Chứa thông tin về đoạn dữ liệu tương ứng

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Một số trường cần quan tâm:
Source port: cổng gửi dữ liệu
Destination port: cổng nhận dữ liệu
Data offset: độ dài TCP header tính bằng số từ 32-bit

1.3. Giao thức TCP

- Các dịch vụ trên nền TCP
  - Rất nhiều dịch vụ chạy trên nền TCP: FTP(21), HTTP(80), SMTP(25), SSH(22), POP3(110), VNC(4899)...
- Sử dụng netcat để kết nối đến một dịch vụ chạy trên nền TCP:

```
nc.exe –v [host] [port]
```

Ví dụ

```
nc.exe –v www.google.com 80
```

1.4. Giao thức UDP

- Giao thức UDP: User Datagram Protocol
  - Cũng là giao thức lõi trong TCP/IP.
  - Cung cấp dịch vụ truyền dữ liệu giữa các ứng dụng.
  - UDP chia nhỏ dữ liệu ra thành các **datagram**
  - Sử dụng trong các ứng dụng khắt khe về mặt thời gian, chấp nhận sai sót: audio, video, game...

1.4. Giao thức UDP

- Đặc tính của UDP
  - Không cần thiết lập kết nối trước khi truyền (Connectionless).
  - Nhanh, chiếm ít tài nguyên dễ xử lý.
  - Hạn chế:
    - Không có cơ chế báo gửi (report).
    - Không đảm báo trật tự các datagram (ordering).
    - Không phát hiện được mất mát hoặc trùng lặp thông tin (loss, duplication).

1.4. Giao thức UDP

- Header của UDP

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Một số trường cần quan tâm:
Source port: cổng gửi dữ liệu
Destination port: cổng nhận dữ liệu
Length: độ dài của gói tin UDP (header luôn có kích thước cố định là 8 bytes)

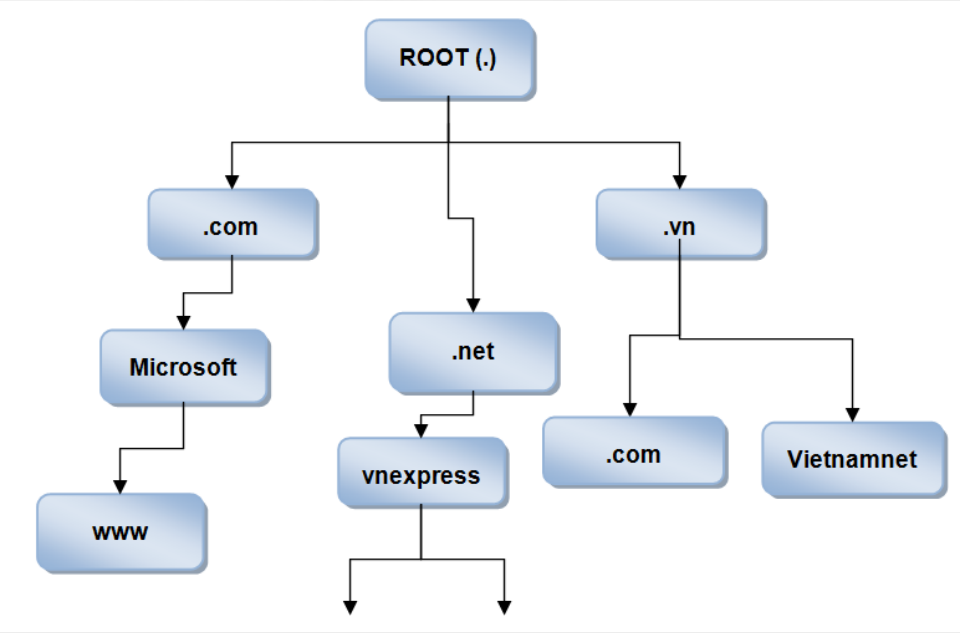
* **Checksum** được sử dụng với mục đích gì?

1.4. Giao thức UDP

- Các dịch vụ trên nền UDP
  - Phân giải tên miền: DNS (53)
  - Streamming: MMS, RTSP...
  - Game

1.5. Hệ thống phân giải tên miền DNS

- Địa chỉ IP khó nhớ với con người.
- DNS – Domain Name System
  - Hệ thống phân cấp làm nhiệm vụ ánh xạ tên miền sang địa chỉ IP và ngược lại.



1.5. Hệ thống phân giải tên miền DNS

- DNS – Domain Name System
  - Các tên miền được phân cấp và quản lý bởi INTERNIC
  - Cấp cao nhất là ROOT, sau đó là cấp 1, cấp 2, ...
  - Thí dụ: **www.hust.edu.vn**

Cấp	**Cấp 4**	**Cấp 3**	**Cấp 2**	**Cấp 1**
Tên miền	www.	hust.	edu.	vn

1.5. Hệ thống phân giải tên miền DNS

- DNS – Domain Name System
  - Tổ chức được cấp tên miền cấp 1 sẽ duy trì cơ sở dữ liệu các tên miền cấp 2 trực thuộc, tổ chức được cấp tên miền cấp 2 sẽ duy trì cơ sở dữ liệu các tên miền cấp 3 trực thuộc...
  - Một máy tính muốn biết địa chỉ của một máy chủ có tên miền nào đó, nó sẽ hỏi máy chủ DNS mà nó nằm trong, nếu máy chủ DNS này không trả lời được nó sẽ chuyển tiếp câu hỏi đến máy chủ DNS cấp cao hơn, DNS cấp cao hơn nếu không trả lời được lại chuyển đến DNS cấp cao hơn nữa...

1.5. Hệ thống phân giải tên miền DNS

- DNS – Domain Name System
  - Việc truy vấn DNS sẽ do hệ điều hành thực hiện.
  - Dịch vụ DNS chạy ở cổng 53 UDP.
  - Công cụ thử nghiệm: **nslookup**
    - Thí dụ: **nslookup www.google.com**

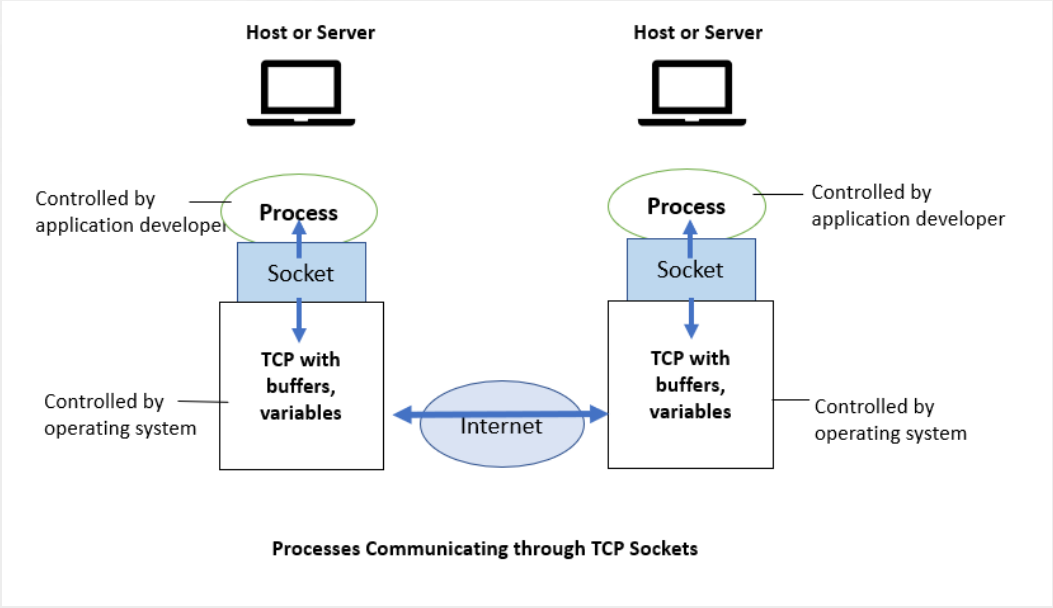
Chương 2. Lập trình Socket

Chương 2. Lập trình socket

2.1. Khái niệm socket
2.2. Giới thiệu Winsock
2.3. Kiến trúc và đặc tính của Winsock
2.4. Lập trình với các hàm cơ bản của WinSock

2.1 Khái niệm socket

- Socket là điểm cuối end-point trong liên kết truyền thông hai chiều (two-way communication) biểu diễn kết nối giữa Client – Server.
- Các lớp Socket được ràng buộc với một cổng port (thể hiện là một con số cụ thể) để các tầng TCP (TCP Layer) có thể định danh ứng dụng mà dữ liệu sẽ được gửi tới.
- Socket là giao diện lập trình mạng được hỗ trợ bởi nhiều ngôn ngữ, hệ điều hành khác nhau.

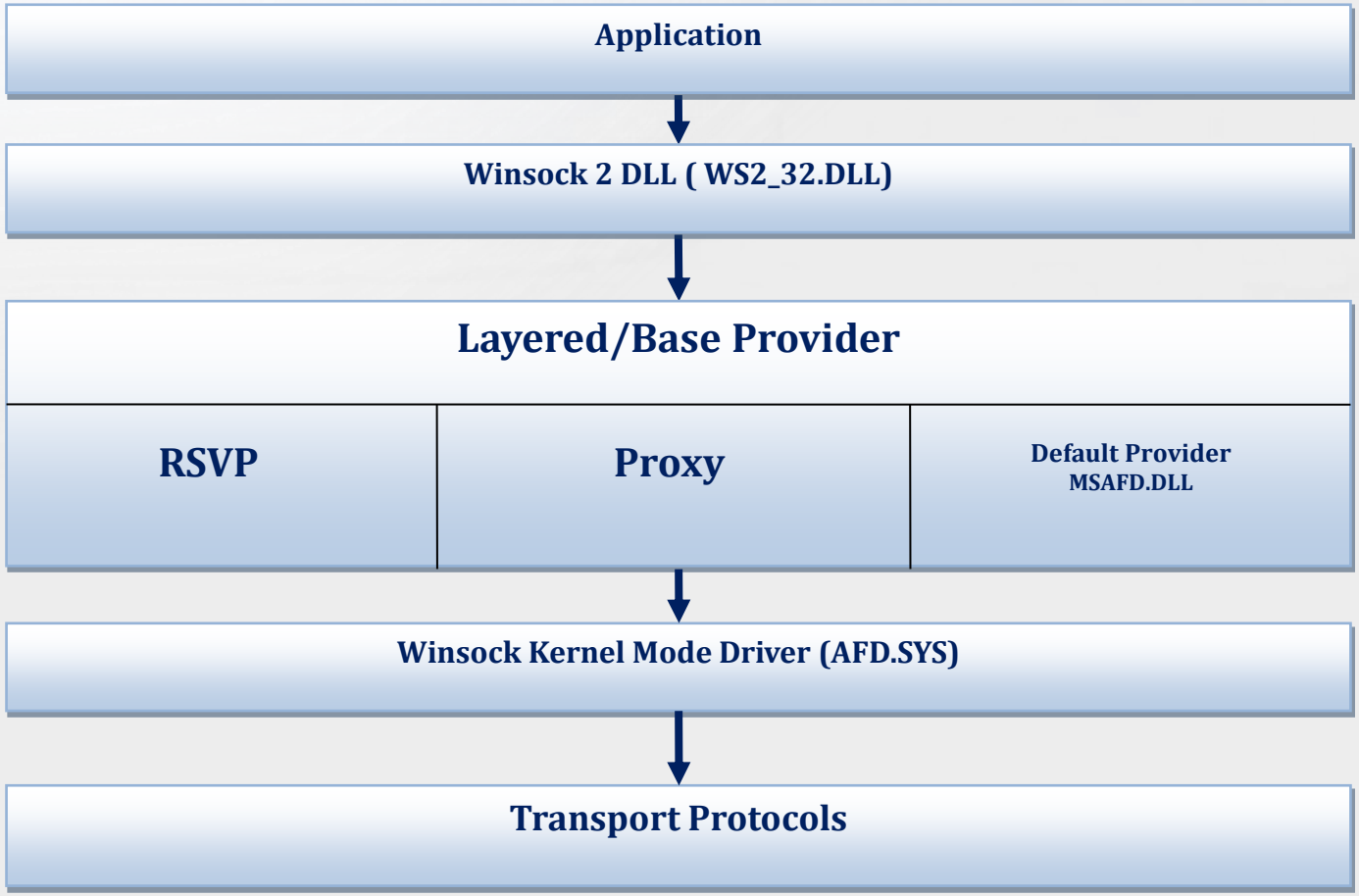


Processes Communicating through TCP Sockets

2.2 Giới thiệu thư viện Winsock

- Windows Socket (WinSock)
  - Bộ thư viện liên kết động của Microsoft.
  - Cung cấp các API dùng để xây dựng ứng dụng mạng hiệu năng cao.

2.3 Kiến trúc và đặc tính của Winsock



2.3.1 Kiến trúc

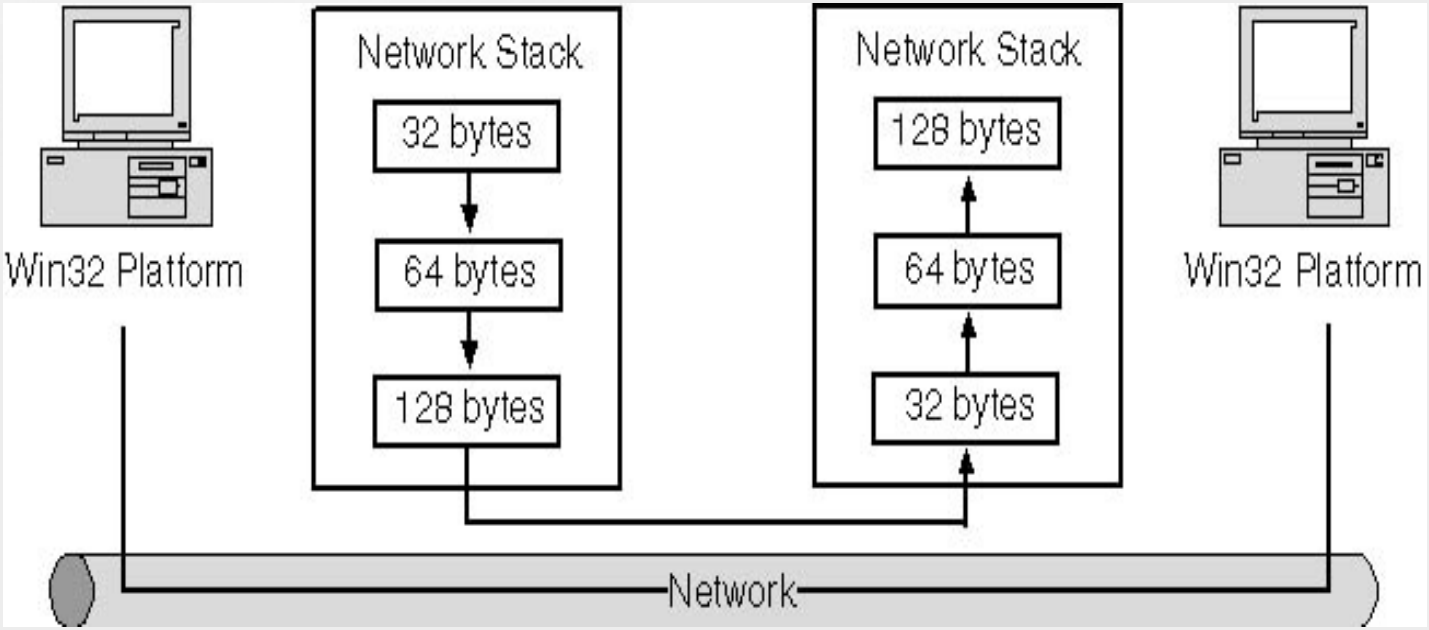
- Các ứng dụng sẽ giao tiếp với thư viện liên kết động ở tầng trên cùng: **WS2_32.DLL.**
- **Provider** do nhà sản xuất của các giao thức cung cấp. Tầng này bổ sung giao thức của các tầng mạng khác nhau cho WinSock như TCP/IP, IPX/SPX, AppleTalk, NetBIOS ... tầng này vẫn chạy ở **UserMode.**
- **WinSock Kernel Mode Driver (AFD.SYS)** là driver chạy ở KernelMode, nhận dữ liệu từ tầng trên, quản lý kết nối, bộ đệm, tài nguyên liên quan đến socket và giao tiếp với driver điều khiển thiết bị.

2.3.1 Kiến trúc

- **Transport Protocols** là các driver ở tầng thấp nhất, điều khiển trực tiếp thiết bị. Các driver này do nhà sản xuất phần cứng xây dựng, và giao tiếp với **AFD.SYS** thông qua giao diện TDI (Transport Driver Interface)

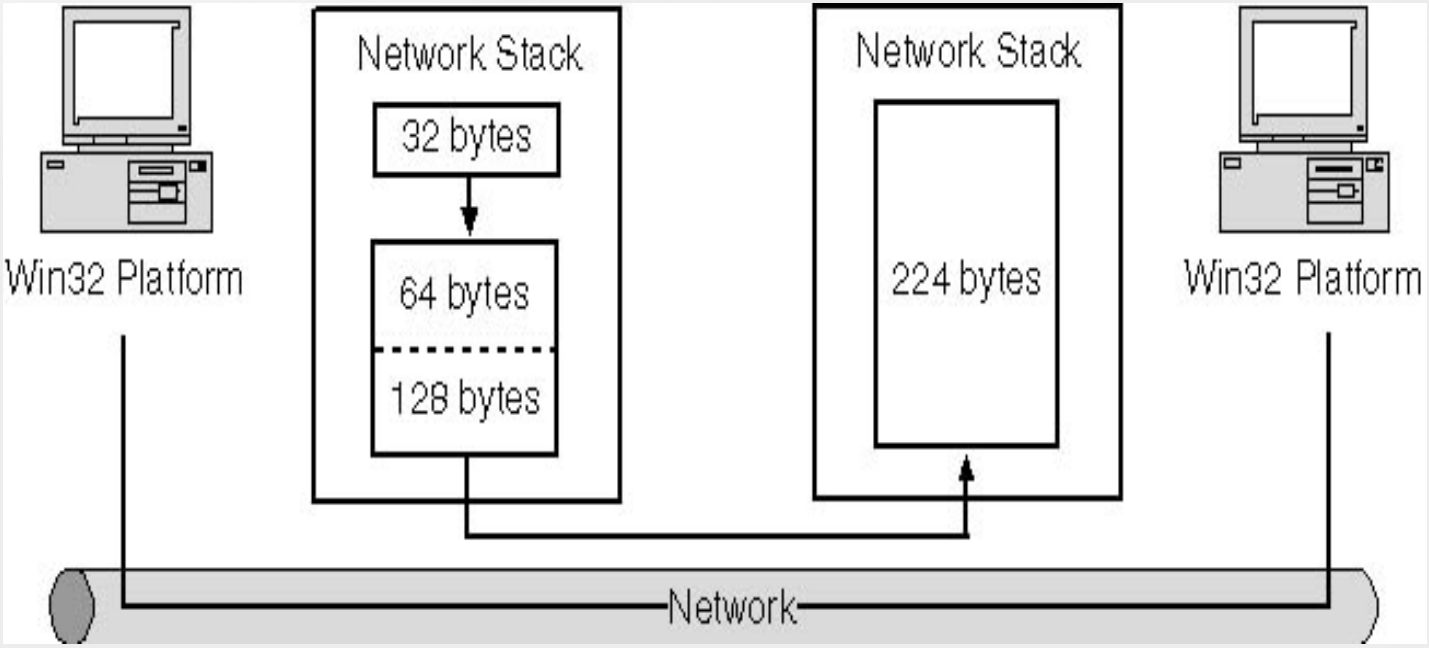
2.3.2 Đặc tính

- Hỗ trợ các giao thức hướng thông điệp (message-oriented)
  - Thông điệp truyền đi được tái tạo nguyên vẹn cả về kích thước và biên ở bên nhận



2.3.2 Đặc tính

- Hỗ trợ các giao thức hướng dòng (stream-oriented)
  - Biên của thông điệp không được bảo toàn khi truyền đi



2.3.2 Đặc tính

- Hỗ trợ các giao thức hướng kết nối và không kết nối
  - Giao thức hướng kết nối (connection oriented) thực hiện thiết lập kênh truyền trước khi truyền thông tin. Thí dụ: TCP
  - Giao thức không kết nối (connectionless) không cần thiết lập kênh truyền trước khi truyền. Thí dụ: UDP

2.3.2 Đặc tính

- Hỗ trợ các giao thức tin cậy và trật tự
  - Tin cậy (reliability): đảm bảo chính xác từng byte được gửi đến đích.
  - Trật tự (ordering): đảm bảo chính xác trật tự từng byte dữ liệu. Byte nào gửi trước sẽ được nhận trước, byte gửi sau sẽ được nhận sau.

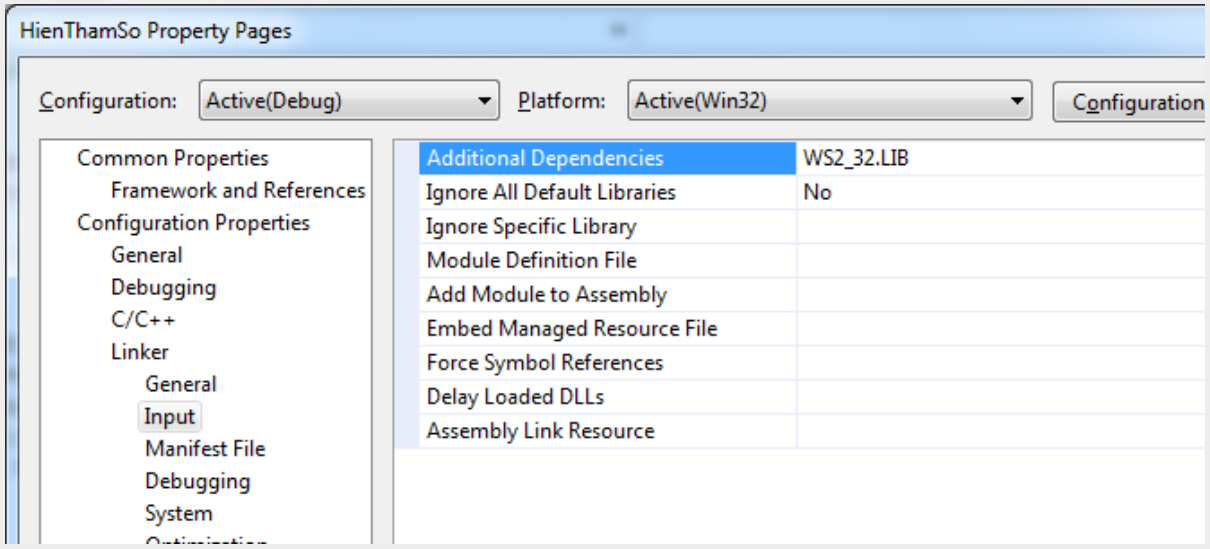
2.3.2 Đặc tính

- Multicast
  - WinSock hỗ trợ các giao thức multicast: gửi dữ liệu đến một hoặc nhiều máy trong mạng.
- Chất lượng dịch vụ - Quality of Service (QoS)
  - Cho phép ứng dụng yêu cầu một phần băng thông dành riêng cho mục đích nào đó. Thí dụ: truyền hình thời gian thực.

2.4 Lập trình Winsock

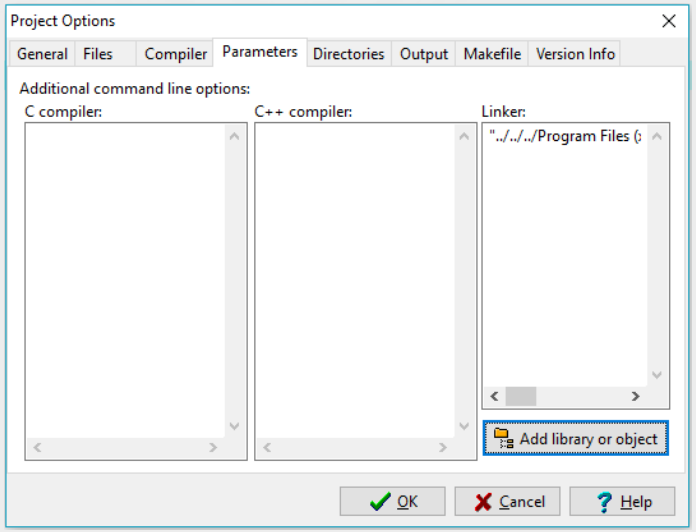
- Chuẩn bị môi trường:
  - Hệ điều hành Windows
  - Công cụ lập trình **Visual Studio**
  - Thêm tiêu đề **WINSOCK2.H** vào đầu mỗi tệp mã nguồn.
  - Thêm thư viện **WS2_32.LIB** vào mỗi Project bằng cách

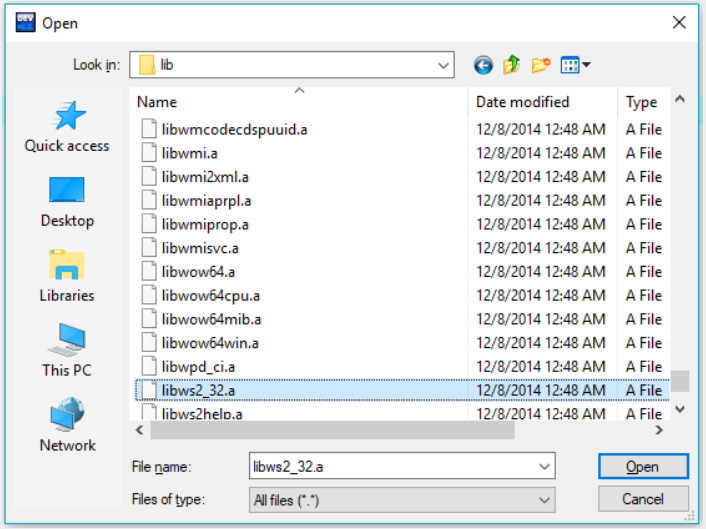
    Project => Property => Configuration Properties=> Linker=>Input=>Additional Dependencies



2.4 Lập trình Winsock

- Chuẩn bị môi trường:
  - Hệ điều hành Windows
  - Công cụ lập trình **Dev-C++**
  - Tạo project mới **File => New => Project**
  - Thêm tiêu đề **WINSOCK2.H** vào đầu mỗi tệp mã nguồn.
  - Thêm thư viện **LIBWS2_32.A** vào mỗi Project bằng cách **Project => Project Options => Parameters => Add library or object** => chọn thư mục **x86_64-w64-mingw32/lib**





2.4 Lập trình Winsock

- Khởi tạo Winsock
  - WinSock cần được khởi tạo ở đầu mỗi ứng dụng trước khi có thể sử dụng
  - Hàm WSAStartup sẽ làm nhiệm vụ khởi tạo

```
int WSAStartup(
    WORD wVersionRequested,
    LPWSADATA lpWSAData
);
```

  - wVersionRequested: [IN] phiên bản WinSock cần dùng.
  - lpWSAData: [OUT] con trỏ chứa thông tin về WinSock cài đặt trong hệ thống.
  - Giá trị trả về:
    - Thành công: 0
    - Thất bại: SOCKET_ERROR

2.4 Lập trình Winsock

- Khởi tạo Winsock
  - Ví dụ:

```
WSADATA wsaData;
WORD wVersion = MAKEWORD(2, 2); // Khởi tạo phiên bản 2.2
if (WSAStartup(wVersion, &wsaData)) {
    printf("Version not supported");
}
```

2.4 Lập trình Winsock

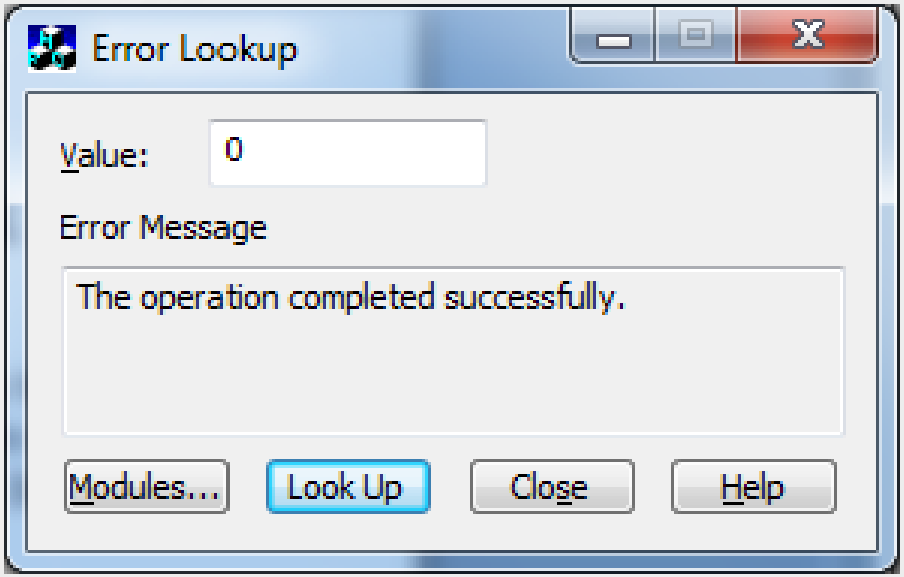
- Giải phóng Winsock
  - Ứng dụng khi kết thúc sử dụng Winsock có thể gọi hàm sau để giải phóng tài nguyên về cho hệ thống

```
int WSACleanup(void);
```

  - Giá trị trả về:
    - Thành công: 0
    - Thất bại: SOCKET_ERROR

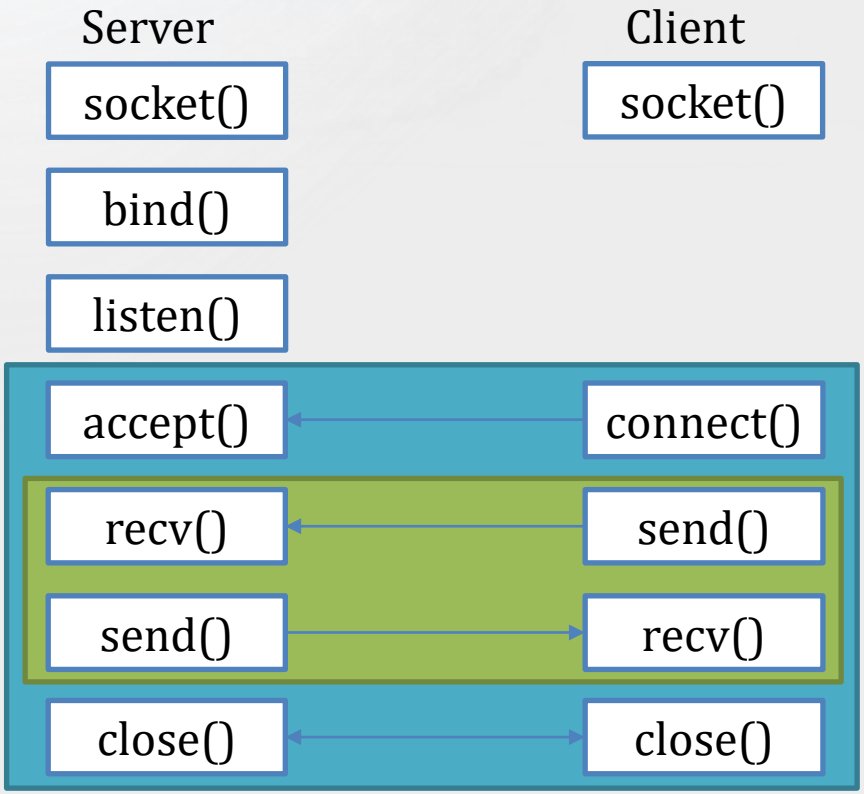
2.4 Lập trình Winsock

- Xác định lỗi
  - Phần lớn các hàm của Winsock nếu thành công đều trả về 0
  - Nếu thất bại, giá trị trả về của hàm là SOCKET_ERROR (-1)
  - Ứng dụng có thể lấy mã lỗi gần nhất bằng hàm
    ```
    int WSAGetLastError(void);
    ```
  - Tra cứu lỗi với công cụ **Error Lookup** trong Visual Studio (menu Tools > Error Lookup)



2.4 Lập trình Winsock

- Giao tiếp giữa server và client thông qua socket



2.4 Lập trình Winsock

- Tạo SOCKET
  - SOCKET là một số nguyên trừu tượng hóa kết nối mạng của ứng dụng.
  - Ứng dụng phải tạo SOCKET trước khi có thể gửi nhận dữ liệu.
  - Hàm **socket** được sử dụng để tạo SOCKET

```
SOCKET socket(int af, int type, int protocol);
```

Trong đó:

- **af**: [IN] Address Family, họ giao thức sẽ sử dụng, thường là AF_INET, AF_INET6.
- **type**: [IN] Kiểu socket, SOCK_STREAM cho TCP/IP và SOCK_DGRAM cho UDP/IP.
- **protocol**: [IN] Giao thức tầng giao vận, IPPROTO_TCP hoặc IPPROTO_UDP

2.4 Lập trình Winsock

- Tạo SOCKET
  - Ví dụ:

```
SOCKET s1, s2; // Khai báo socket s1,s2

// Tạo socket TCP
s1 = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// Tạo socket UDP
s2 = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

2.4 Lập trình Winsock

- Xác định địa chỉ
  - Winsock sử dụng **sockaddr_in** để lưu địa chỉ của ứng dụng đích cần nối đến.
  - Ứng dụng cần khởi tạo thông tin trong cấu trúc này

```
struct sockaddr_in {
    short sin_family;          // Họ giao thức, thường là AF_INET
    u_short sin_port;          // Cổng, dạng big-endian
    struct in_addr sin_addr; // Địa chỉ IP
    char sin_zero[8];          // Không sử dụng với IPv4
};
```

2.4 Lập trình Winsock

- Xác định địa chỉ
  - Sử dụng các hàm hỗ trợ :
    - Chuyển đổi địa chỉ IP dạng xâu sang số nguyên 32 bit

```
unsigned long inet_addr(const char FAR *cp);
```

    - Chuyển đổi địa chỉ từ dạng **in_addr** sang dạng xâu

```
char FAR *inet_ntoa(struct in_addr in);
```

    - Chuyển đổi little-endian => big-endian (network order)

```
// Chuyển đổi 4 byte từ little-endian=>big-endian
u_long htonl(u_long hostlong)
// Chuyển đổi 2 byte từ little-endian=>big-endian
u_short htons(u_short hostshort)
```

    - Chuyển đổi big-endian => little-endian (host order)

```
// Chuyển 4 byte từ big-endian=>little-endian
u_long ntohl(u_long netlong)
// Chuyển 2 byte từ big-endian=>little-endian
u_short ntohs(u_short netshort)
```

2.4 Lập trình Winsock

- Xác định địa chỉ
  - Ví dụ: gán địa chỉ 192.168.0.10:8000 vào cấu trúc sockaddr_in

```
// Khai báo biến lưu địa chỉ của server
SOCKADDR_IN addr;

// Họ địa chỉ IPV4
addr.sin_family = AF_INET;
// Chuyển xâu địa chỉ IP sang số 4 byte dạng network-byte order
// rồi gán cho trường sin_addr
addr.sin_addr.s_addr = inet_addr("192.168.0.10");
// Chuyển đổi cổng sang dạng network-byte order
// rồi gán cho trường sin_port
addr.sin_port = htons(8000);
```

2.4 Lập trình Winsock

- Phân giải tên miền
  - Đôi khi địa chỉ của máy đích được cho dưới dạng tên miền
  - Ứng dụng cần thực hiện phân giải tên miền để có địa chỉ thích hợp
  - Hàm **getnameinfo** và **getaddrinfo** sử dụng để phân giải tên miền
  - Cần thêm tệp tiêu đề WS2TCPIP.H

```
int getaddrinfo(
    const char* nodename,  // Tên miền hoặc địa chỉ cần phân giải
    const char* servname,  // Dịch vụ hoặc cổng
    const struct addrinfo* hints, // Cấu trúc gợi ý
    struct addrinfo** res          // Kết quả
);
```

  - Giá trị trả về
    - Thành công: 0
    - Thất bại: mã lỗi
  - Giải phóng: **freeaddrinfo()**

2.4 Lập trình Winsock

- Phân giải tên miền
  - Cấu trúc **addrinfo**: danh sách liên kết đơn chứa thông tin về tên miền tương ứng

```
struct addrinfo {
    int ai_flags;              // Thường là AI_CANONNAME
    int ai_family;             // Thường là AF_INET
    int ai_socktype;           // Loại socket
    int ai_protocol;           // Giao thứ giao vận
    size_t ai_addrlen;         // Chiều dài của ai_addr
    char* ai_canonname;        // Tên miền
    struct sockaddr* ai_addr; // Địa chỉ socket đã phân giải
    struct addrinfo* ai_next; // Con trỏ tới cấu trúc sau
};
```

2.4 Lập trình Winsock

- Phân giải tên miền
  - Đoạn chương trình sau sẽ thực hiện phân giải địa chỉ cho tên miền hust.edu.vn

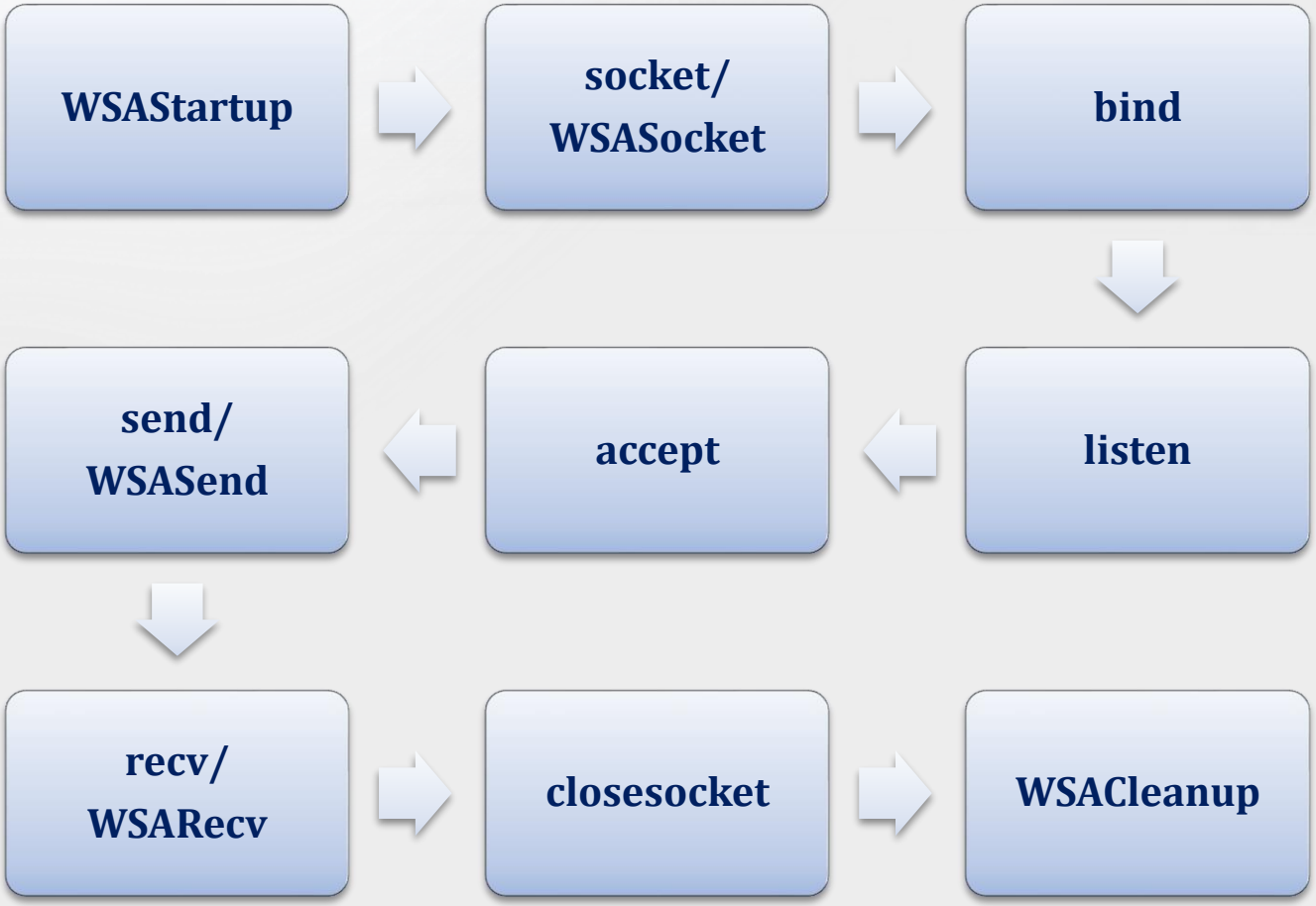
```
// Phan giai ten mien
addrinfo* info;
SOCKADDR_IN addr;
int ret = getaddrinfo("hust.edu.vn", "http", NULL, &info);

// Neu phan giai thanh cong thi sao chep dia chi vao bien addr
if (ret == 0)
   memcpy(&addr, info->ai_addr, info->ai_addrlen);
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Việc truyền nhận dữ liệu sử dụng giao thức TCP sẽ bao gồm hai phần: ứng dụng phía client và phía server.
  - **Ứng dụng phía server:**
    - Khởi tạo WinSock qua hàm **WSAStartup**
    - Tạo SOCKET qua hàm **socket** hoặc **WSASocket**
    - Gắn SOCKET vào một giao diện mạng thông qua hàm **bind**
    - Chuyển SOCKET sang trạng thái đợi kết nối qua hàm **listen**
    - Chấp nhận kết nối từ client thông qua hàm **accept**
    - Gửi dữ liệu tới client thông qua hàm **send** hoặc **WSASend**
    - Nhận dữ liệu từ client thông qua hàm **recv** hoặc **WSARecv**
    - Đóng SOCKET khi việc truyền nhận kết thúc bằng hàm **closesocket**
    - Giải phóng WinSock bằng hàm **WSACleanup**

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)



2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP – Ứng dụng phía server (tiếp)
  - Hàm **bind**: gắn SOCKET vào một giao diện mạng của máy

```
int bind(SOCKET s,
         const struct sockaddr FAR* name,
         int namelen);
```

  - Trong đó:
    - **s: [IN]** SOCKET vừa được tạo bằng hàm socket
    - **name: [IN]** địa chỉ của giao diện mạng cục bộ
    - **namelen: [IN]** chiều dài của cấu trúc name
  - Ví dụ:

```
// Khai bao dia chi cua server
SOCKADDR_IN addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(9000);

bind(listener, (SOCKADDR*)&addr, sizeof(addr));
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Hàm **listen**: chuyển SOCKET sang trạng thái đợi kết nối

    ```
    int listen(SOCKET s, int backlog);
    ```
  - Trong đó:
    - **s**: [IN] SOCKET đã được tạo trước đó bằng socket/WSASocket
    - **backlog**: [IN] chiều dài hàng đợi chấp nhận kết nối

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Hàm **accept**: chấp nhận kết nối

```
SOCKET accept(SOCKET s,
        struct sockaddr FAR* addr,
        int FAR* addrlen);
```

  - Trong đó:
    - **s**: **[IN]** SOCKET hợp lệ, đã được bind và listen trước đó
    - **addr: [OUT]** địa chỉ của client kết nối đến
    - **addrlen: [IN/OUT]** con trỏ tới chiều dài của cấu trúc addr. Ứng dụng cần khởi tạo addrlen trỏ tới một số nguyên chứa chiều dài của addr
  - Giá trị trả về là một SOCKET mới, sẵn sàng cho việc gửi nhận dữ liệu trên đó. Ứng với mỗi kết nối của client sẽ có một SOCKET riêng.

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Ví dụ hàm **accept**

```
// s là socket đã được khởi tạo để chờ các kết nối

SOCKET s1 = accept(s, NULL, NULL);
// s1 là socket đại diện cho kết nối giữa server và client1
// trong trường hợp này không cần quan tâm đến địa chỉ của client1

SOCKADDR_IN clientAddr;
int clientAddrLen = sizeof(clientAddr);
SOCKET s2 = accept(s, (SOCKADDR*)&clientAddr, &clientAddrLen);
// s2 là socket đại diện cho kết nối giữa server và client2
// clientAddr chứa dữ liệu địa chỉ của client2
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Hàm **send**: gửi dữ liệu trên SOCKET

```
int send(SOCKET s,
         const char FAR * buf,
         int len,
         int flags);
```

  - Trong đó:
    - **s**: **[IN]** SOCKET hợp lệ, đã được accept trước đó
    - **buf: [IN]** địa chỉ của bộ đệm chứa dữ liệu cần gửi
    - **len: [IN]** số byte cần gửi
    - **flags:[IN]** cờ quy định cách thức gửi, có thể là 0, MSG_OOB, MSG_DONTROUTE
  - Giá trị trả về:
    - Thành công: số byte gửi được, có thể nhỏ hơn **len**
    - Thất bại: SOCKET_ERROR

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Ví dụ hàm **send**:

```
// client là socket đã được chấp nhận bởi server
char* str = "Hello Network Programming";
int res = send(client, str, strlen(str), 0);
if (res != SOCKET_ERROR)
printf("%d bytes are sent", res);

char buf[256];
for (int i = 0; i < 10; i++)
    buf[i] = i;
res = send(client, buf, 10, 0);

long l = 1234; 4 byte
send(client, &l, sizeof(l), 0);
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Hàm **recv**: nhận dữ liệu trên SOCKET

```
int recv(SOCKET s,
         const char FAR * buf,
         int len,
         int flags);
```

  - Trong đó
    - **s**: **[IN]** SOCKET hợp lệ, đã được accept trước đó
    - **buf: [OUT]** địa chỉ của bộ đệm nhận dữ liệu
    - **len: [IN]** kích thước bộ đệm
    - **flags:[IN]** cờ quy định cách thức nhận, có thể là 0, MSG_PEEK, MSG_OOB, MSG_WAITALL
  - Giá trị trả về
    - Thành công: số byte nhận được, có thể nhỏ hơn **len**
    - Thất bại: SOCKET_ERROR

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Ví dụ hàm **recv**:

```
// client là socket đã được chấp nhận bởi server
char buf[256];
int res = recv(client, buf, sizeof(buf), 0);

while (true) {
    res = recv(client, buf, sizeof(buf), 0);
    if (res <= 0)
        break;
    // process buffer
}
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía server (tiếp)
  - Hàm **closesocket**: đóng kết nối trên một socket

    ```
    int closesocket(SOCKET s);
    ```
  - Trong đó
    - **s**: **[IN]** SOCKET hợp lệ, đã kết nối
  - Giá trị trả về
    - Thành công: 0
    - Thất bại: SOCKET_ERROR

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ví dụ minh họa

```
#include <stdio.h>
#include <winsock2.h>

int main()
{
    // Khoi tao thu vien Winsock
    WSADATA wsa;
    WSAStartup(MAKEWORD(2, 2), &wsa);

    // Tao doi tuong socket
    SOCKET listener = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    // Khai bao dia chi cua server
    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(9000);
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ví dụ minh họa (tiếp)

```
// Chuyen sang sang thai cho ket noi
bind(listener, (SOCKADDR*)&addr, sizeof(addr));
listen(listener, 5);

// Chap nhan ket noi
SOCKET client = accept(listener, NULL, NULL);

char msg[256];

// Nhan cau chao tu client
int ret = recv(client, msg, sizeof(msg), 0);

if (ret <= 0)
{
    printf("Loi ket noi!");
    system("pause");
    return 1;
}
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ví dụ minh họa (tiếp)

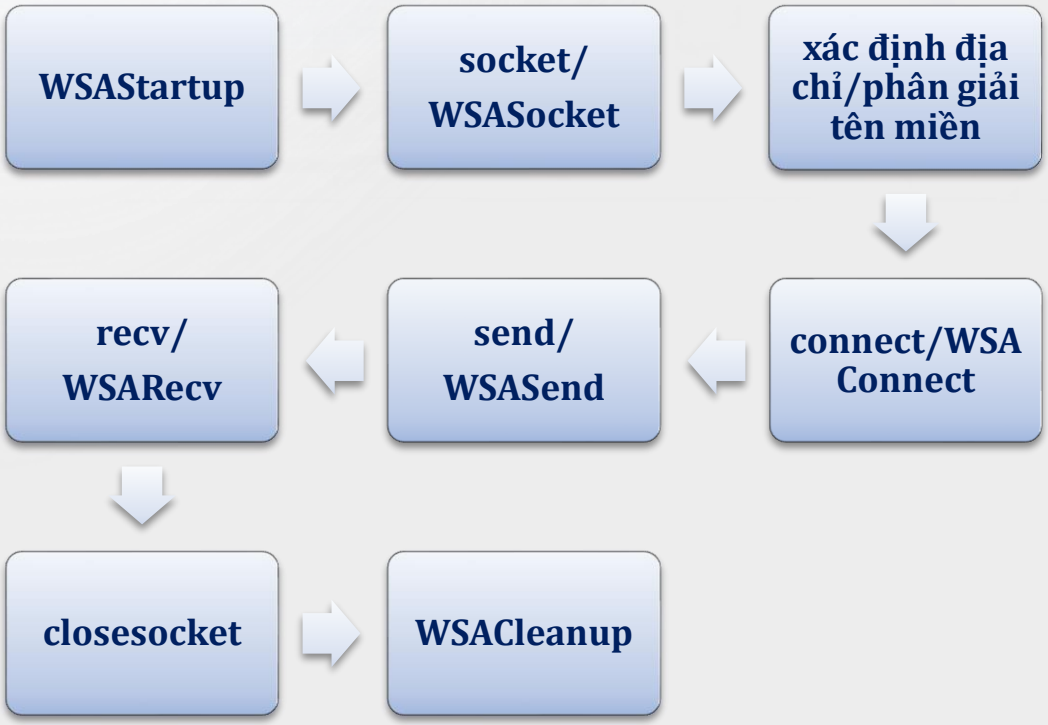
```
    // Them ky tu ket thuc xau va in ra man hinh
    if (ret < 256) msg[ret] = 0;
    printf("Received: %s\n", msg);

    // Lien tuc nhap chuoi ky tu tu ban phim va gui sang client
    while (1)
    {
        printf("Nhap xau ky tu: ");
        fgets(msg, sizeof(msg), stdin);
        if (strcmp(msg, "exit") == 0) break;
        send(client, msg, strlen(msg), 0);
    }

    closesocket(client);
    closesocket(listener);
    WSACleanup();

    return 0;
}
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía client
    - Khởi tạo WinSock qua hàm **WSAStartup**
    - Tạo SOCKET qua hàm **socket** hoặc **WSASocket**
    - Điền thông tin về server vào cấu trúc **sockaddr_in**
    - Kết nối tới server qua hàm **connect** hoặc **WSAConnect**
    - Gửi dữ liệu tới server thông qua hàm **send** hoặc **WSASend**
    - Nhận dữ liệu từ server thông qua hàm **recv** hoặc **WSARecv**
    - Đóng SOCKET khi việc truyền nhận kết thúc bằng hàm **closesocket**
    - Giải phóng WinSock bằng hàm **WSACleanup**

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía client (tiếp)



2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP - Ứng dụng phía client (tiếp)
  - Địa chỉ của server xác định trong cấu trúc **sockaddr_in** nhờ hàm **inet_addr** hoặc theo **getaddrinfo**
  - Hàm **connect**: kết nối đến server

```
int connect(SOCKET s,
      const struct sockaddr FAR* name,
      int namelen);
```

  - Trong đó
    - **s: [IN]** SOCKET đã được tạo bằng **socket** hoặc **WSASocket** trước đó
    - **name:[IN]** địa chỉ của server
    - **namelen:[IN]** chiều dài cấu trúc **name**
  - Giá trị trả về
    - Thành công: 0
    - Thất bại: SOCKET_ERROR

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ví dụ minh họa

```
#include <stdio.h>
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <winsock2.h>

int main()
{
    // Khoi tao thu vien Winsock
    WSADATA wsa;
    WSAStartup(MAKEWORD(2, 2), &wsa);

    // Tao doi tuong socket
    SOCKET client = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    // Khai bao dia chi cua server
    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    addr.sin_port = htons(9000);
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ví dụ minh họa (tiếp)

```
// Ket noi den server
int ret = connect(client, (SOCKADDR*)&addr, sizeof(addr));
if (ret == SOCKET_ERROR)
{
    printf("Loi ket noi!");
    system("pause");
    return 1;
}

// Gui cau chao len server
char msg[256] = "Hello server! I am a new client.";
send(client, msg, strlen(msg), 0);
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng TCP
  - Ví dụ minh họa (tiếp)

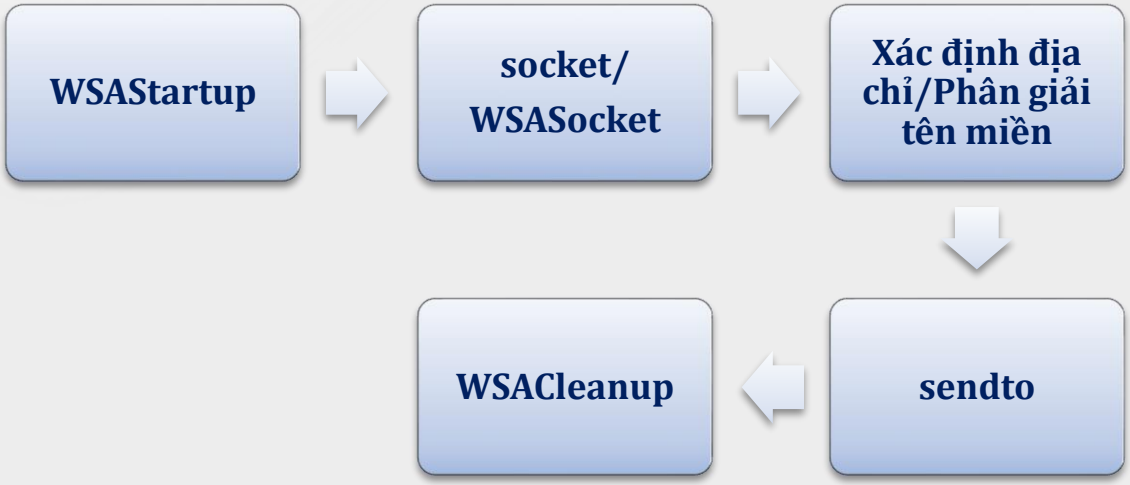
```
// Lien tuc nhan thong diep tu server va in ra man hinh
while (1)
{
    ret = recv(client, msg, sizeof(msg), 0);
    if (ret <= 0)
    {
        printf("Loi ket noi!");
        break;
    }

    // Them ky tu ket thuc xau va in ra man hinh
    if (ret < 256) msg[ret] = 0;
    printf("Received: %s\n", msg);
}

closesocket(client);
WSACleanup();
return 0;
```

2.4 Lập trình Winsock

- Ví dụ minh họa
  - Tạo client gửi thông điệp đến netcat server
  - Tạo client gửi lệnh GET đến website (sử dụng HTTP) và hiển thị kết quả trả về **=> Bài tập Download File**
  - Tạo client và server truyền dữ liệu là chuỗi ký tự
  - Tạo client và server truyền dữ liệu là số
  - Tạo client và server truyền dữ liệu là file

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP
  - Giao thức UDP là giao thức không kết nối (Connectionless)
  - Ứng dụng không cần phải thiết lập kết nối trước khi gửi tin.
  - Ứng dụng có thể nhận được tin từ bất kỳ máy tính nào trong mạng.
  - Trình tự gửi thông tin ở bên gửi như sau



2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP - Ứng dụng bên gửi
  - Hàm **sendto**: gửi dữ liệu đến một máy tính bất kỳ

```
int sendto(
   SOCKET s, // socket đã tạo bằng hàm socket
   const char FAR* buf, // bộ đệm chứa dữ liệu cần gửi
   int len, // số byte cần gửi
   int flags, // cờ, tương tự như hàm send
   const struct sockaddr FAR* to, // địa chỉ đích
   int tolen // chiều dài địa chỉ đích
);
```

  - Giá trị trả về
    - Thành công: số byte gửi được, có thể nhỏ hơn **len**
    - Thất bại: SOCKET_ERROR

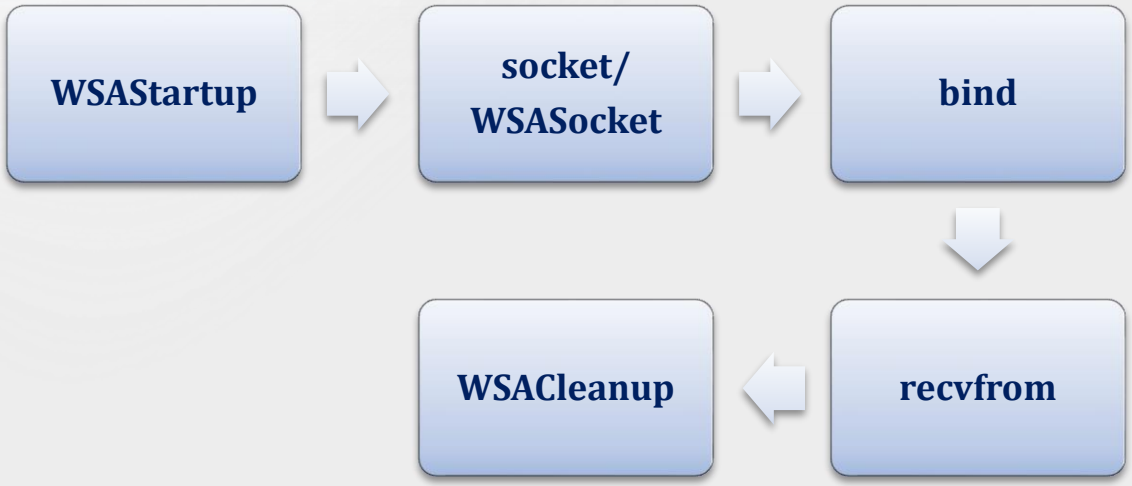
2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP
  - Đoạn chương trình sau sẽ gửi một xâu tới địa chỉ 202.191.56.69:8888

```
char buf[] = "Hello Network Programming"; // Xâu cần gửi
SOCKET sender; // SOCKET để gửi
SOCKADDR_IN receiverAddr; // Địa chỉ nhận
// Tạo socket để gửi tin
sender = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
// Điền địa chỉ đích
receiverAddr.sin_family = AF_INET;
receiverAddr.sin_port = htons(8888);
receiverAddr.sin_addr.s_addr = inet_addr("202.191.56.69");
// Thực hiện gửi tin
sendto(sender, buf, strlen(buf), 0, (SOCKADDR*)&receiverAddr,
sizeof(receiverAddr));
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP
  - Trình tự nhận thông tin ở bên nhận như sau



2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP - Ứng dụng bên nhận
  - Hàm **recvfrom**: nhận dữ liệu từ một socket

```
int recvfrom(
   SOCKET s,           // SOCKET sẽ nhận dữ liệu
   char FAR* buf,      // địa chỉ bộ đệm chứa dữ liệu sẽ nhận được
   int len,            // kích thước bộ đệm
   int flags,          // cờ, tương tự như hàm recv
   struct sockaddr FAR* from,   // địa chỉ của bên gửi
   int FAR* fromlen // chiều dài cấu trúc địa chỉ của bên
                    // gửi, khởi tạo là chiều dài của from
);
```

  - Giá trị trả về
    - Thành công: số byte nhận được
    - Thất bại: SOCKET_ERROR

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP
  - Đoạn chương trình sau sẽ nhận đữ liệu datagram từ cổng 8888 và hiển thị ra màn hình

```
SOCKET receiver;
SOCKADDR_IN addr, source;
int len = sizeof(source);
// Tạo socket UDP
receiver = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
// Khởi tạo địa chỉ và cổng 8888
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(8888); // Đợi UDP datagram ở cổng 8888

// Bind socket vào tất cả các giao diện và cổng 8888
bind(receiver, (sockaddr*)&addr, sizeof(SOCKADDR_IN));
```

2.4 Lập trình Winsock

- Truyền dữ liệu sử dụng UDP

```
// Lặp đợi gói tin
while (1) {
    // Nhận dữ liệu từ mạng
    datalen = recvfrom(receiver, buf, 100, 0, (sockaddr*)&source,
&len);
    // Kiểm tra chiều dài
    if (datalen > 0) {
        buf[datalen] = 0;
        printf("Data:%s", buf); // Hiển thị ra màn hình
    }
}
```

Câu hỏi:

1. Khi đóng kết nối, recvfrom() có trả về giá trị lỗi không?
2. Nếu dữ liệu được gửi trong 1 lệnh lớn hơn buffer bên nhận thì hiện tượng gì sẽ xảy ra?
3. Chương trình trên có thể nhận dữ liệu từ nhiều kết nối khác nhau không?

2.4 Lập trình Winsock

- Sử dụng Netcat để gửi nhận dữ liệu đơn giản
  - Netcat là một tiện ích mạng rất đa năng.
  - Có thể sử dụng như TCP server:

    nc.exe -v -l -p <*cổng đợi kết nối*>

    Ví dụ: **nc.exe -l -p 8888**
  - Có thể sử dụng như TCP client:

    nc.exe -v <*ip/tên miền*> <*cổng*>

    Ví dụ: **nc.exe 127.0.0.1 80**
  - Sử dụng như UDP receiver:

    nc.exe -v -l -u -p <*cổng đợi kết nối*>

    Ví dụ: **nc.exe -v -l -u -p 8888**
  - Sử dụng như UDP sender:

    nc.exe -v -u <*ip/tên miền*> <*cổng*>

    Ví dụ: **nc.exe -v -u 192.168.0.1 80**

2.4 Lập trình Winsock

- Một số hàm khác
  - **getpeername**: lấy địa chỉ đầu kia mà SOCKET kết nối đến

```
int getpeername(
    SOCKET s,   // SOCKET cần lấy địa chỉ
    struct sockaddr FAR* name, // địa chỉ lấy được
    int FAR* namelen           // chiều dài địa chỉ
);
```

  - **getsockname**: lấy địa chỉ cục bộ của SOCKET

```
int getsockname(
    SOCKET s,               // SOCKET cần lấy địa chỉ
    struct sockaddr FAR* name,  // địa chỉ lấy được
    int FAR* namelen            // chiều dài địa chỉ
);
```

Bài tập

1. Viết chương trình TCPClient, kết nối đến một máy chủ xác định bởi tên miền hoặc địa chỉ IP. Sau đó nhận dữ liệu từ bàn phím và gửi đến server. Tham số được truyền vào từ dòng lệnh có dạng

TCPClient.exe <Địa chỉ IP/Tên miền> <Cổng>

2. Viết chương trình TCPServer, đợi kết nối ở cổng xác định bởi tham số dòng lệnh. Mỗi khi có client kết nối đến, thì gửi xâu chào được chỉ ra trong một tệp tin xác định, sau đó ghi toàn bộ nội dung client gửi đến vào một tệp tin khác được chỉ ra trong tham số dòng lệnh

TCPServer.exe <Cổng> <Tệp tin chứa câu chào> <Tệp tin lưu nội dung client gửi đến>

VD: TCPServer.exe 8888 chao.txt client.txt

Bài tập

3. Viết chương trình **clientinfo** thực hiện kết nối đến một máy chủ xác định và gửi thông tin về tên máy, danh sách các ổ đĩa có trong máy, kích thước các ổ đĩa. Địa chỉ (tên miền) và cổng nhận vào từ tham số dòng lệnh.

VD: clientinfo.exe localhost 1234

4. Viết chương trình **serverinfo** đợi kết nối từ **clientinfo** và thu nhận thông tin từ client, hiện ra màn hình. Tham số dòng lệnh truyền vào là cổng mà serverinfo sẽ đợi kết nối

VD: serverinfo.exe 1234

Chương 3. Giới thiệu lập trình đa luồng

Chương 3. Giới thiệu lập trình đa luồng

3.1. Khởi tạo và thực thi các luồng trên Windows
3.2. Đồng bộ và tránh xung đột trong lập trình đa luồng

3.1 Khởi tạo và thực thi các luồng

Khởi tạo luồng mới:

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES ThreadAttributes,
    DWORD StackSize,
    LPTHREAD_START_ROUTINE StartAddress,
    LPVOID Parameter,
    DWORD CreationFlags,
    LPDWORD ThreadId);
```

Các tham số cần quan tâm:

- **StartAddress** tên của hàm thực thi, cần được khai báo trước
- **Parameter** con trỏ tham số truyền vào hàm thực thi

Kết quả trả về:

- **FALSE** – nếu xảy ra lỗi, có thể dùng hàm GetLastError() để xác định
- **NOT FALSE** – HANDLE sử dụng để tham chiếu đến luồng

3.1 Khởi tạo và thực thi các luồng

Khởi tạo luồng mới:

- Hàm CreateThread() yêu cầu khai báo hàm thực thi (có thể khai báo prototype trước khi thực hiện nội dung hàm)
- Hàm thực thi được chạy ngay sau khi luồng được tạo
- Ví dụ khai báo prototype của hàm thực thi:

```
DWORD WINAPI MyThreadStart(LPVOID p);
```

3.1 Khởi tạo và thực thi các luồng

Xóa luồng:

- Mục đích giải phóng tài nguyên (bộ nhớ) sau khi các luồng thực hiện xong.
- Nếu tạo quá nhiều luồng mà không giải phóng tài nguyên:
  - Gây rò rỉ bộ nhớ
  - Không tạo thêm được luồng mới
- Khi chương trình kết thúc, các luồng được tự động giải phóng.

```
BOOL CloseHandle(HANDLE hObject);
```

3.1 Khởi tạo và thực thi các luồng

Ví dụ:

```
#include <stdio.h>
#include <windows.h>
DWORD WINAPI helloFunc(LPVOID arg) {
    printf("Hello Thread\n");
    return 0;
}
main() {
    HANDLE hThread = CreateThread(NULL, 0, helloFunc,
        NULL, 0, NULL);
}
```

Kết quả thực hiện chương trình?

3.1 Khởi tạo và thực thi các luồng

Ví dụ:

Một trong hai khả năng xảy ra:

- Dòng chữ “Hello Thread” được in ra màn hình
- Dòng chữ không được in ra màn hình => Do chương trình kết thúc trước khi hàm thực thi chạy

=> Cần có cơ chế chờ cho luồng chạy xong

3.1 Khởi tạo và thực thi các luồng

Ví dụ:

```
#include <stdio.h>
#include <windows.h>
BOOL threadDone = FALSE;
DWORD WINAPI helloFunc(LPVOID arg) {
    printf("Hello Thread\n");
    threadDone = TRUE;
    return 0;
}
main() {
    HANDLE hThread = CreateThread(NULL, 0, helloFunc, NULL, 0,
        NULL);
    while (!threadDone);
}
```

Vấn đề với đoạn chương trình?

3.1 Khởi tạo và thực thi các luồng

Cơ chế đợi luồng thực thi:
Sử dụng hàm **WaitForSingleObject()** để đợi 1 luồng thực thi xong.

```
DWORD WaitForSingleObject(
          HANDLE hHandle,
          DWORD dwMilliseconds );
```

Luồng gọi hàm sẽ dừng và đợi cho đến khi:
- Hết giờ sau **dwMilliseconds** giây
- Luồng thực thi xong

Nếu chỉ muốn chờ cho đến khi hàm thực thi xong thì truyền **INFINITE** cho tham số **dwMilliseconds**.

3.1 Khởi tạo và thực thi các luồng

Ví dụ:

```
#include <stdio.h>
#include <windows.h>
// BOOL threadDone = FALSE;
DWORD WINAPI helloFunc(LPVOID arg) {
    printf("Hello Thread\n");
    // threadDone = TRUE;
    return 0;
}
main() {
    HANDLE hThread = CreateThread(NULL, 0, helloFunc, NULL, 0,
         NULL);
    WaitForSingleObject(hThread, INFINITE);
}
```

3.1 Khởi tạo và thực thi các luồng

Cơ chế đợi luồng thực thi:
Sử dụng hàm **WaitForMultipleObject()** để đợi 1 hoặc nhiều luồng thực thi xong (tối đa 64 luồng).

```
DWORD WaitForMultipleObjects(
      DWORD nCount,
      CONST HANDLE *lpHandles, // array
      BOOL fWaitAll, // wait for one or all
      DWORD dwMilliseconds);
```

nCount là số phần tử trong mảng lpHandles
fWaitAll = TRUE => hàm trả về kết quả nếu tất cả các luồng thực hiện xong
fWaitAll = FALSE => hàm trả về kết quả nếu một trong các luồng thực hiện xong, giá trị trả về là chỉ số của luồng trong mảng

3.1 Khởi tạo và thực thi các luồng

Ví dụ:

```
#include <stdio.h>
#include <windows.h>
const int numThreads = 4;

DWORD WINAPI helloFunc(LPVOID arg) {
    printf("Hello Thread\n");
    return 0;
}

main() {
    HANDLE hThread[numThreads];
    for (int i = 0; i < numThreads; i++)
        hThread[i] = CreateThread(NULL, 0, helloFunc, NULL, 0, NULL);
    WaitForMultipleObjects(numThreads, hThread, TRUE, INFINITE);
}
```

3.1 Khởi tạo và thực thi các luồng

Ví dụ: cập nhật đoạn chương trình để hiển thị các thông điệp ứng với luồng được tạo.

```
Hello from Thread #0
Hello from Thread #1
Hello from Thread #2
Hello from Thread #3
```

3.1 Khởi tạo và thực thi các luồng

Ví dụ: cập nhật đoạn chương trình để hiển thị các thông điệp ứng với luồng được tạo.

```
DWORD WINAPI threadFunc(LPVOID pArg) {
    int* p = (int*)pArg;
    int myNum = *p;
    printf("Thread number % d\n", myNum);
}
. . .
// from main():
for (int i = 0; i < numThreads; i++) {
    hThread[i] = CreateThread(NULL, 0, threadFunc, &params[i], 0, NULL);
}
```

3.2 Đồng bộ và tránh xung đột trong lập trình đa luồng

- Việc truy nhập đồng thời vào cùng một biến từ nhiều luồng sẽ dẫn đến các xung đột
  - Xung đột đọc/ghi dữ liệu
  - Xung đột ghi/ghi dữ liệu
- Lỗi phổ biến trong lập trình đa luồng
- Có thể không rõ ràng ở tất cả các tình huống => Khó gỡ lỗi

3.2 Đồng bộ và tránh xung đột trong lập trình đa luồng

- Các phương pháp tránh xung đột
  - Hạn chế sử dụng biến toàn cục, nên sử dụng biến cục bộ khai báo trong hàm thực thi của luồng.
  - Quản lý việc truy nhập các tài nguyên dùng chung
    - Mutex
    - **Critical Section**
    - Events
    - Semaphores

3.2 Đồng bộ và tránh xung đột trong lập trình đa luồng

- Sử dụng **Critical Section**
  - Là cơ chế đơn giản, được sử dụng nhiều nhất
  - Tạo đối tượng mới:

```
CRITICAL_SECTION cs;
```

  - Khởi tạo và hủy đối tượng

```
InitializeCriticalSection(&cs);
DeleteCriticalSection(&cs);
```

3.2 Đồng bộ và tránh xung đột trong lập trình đa luồng

- Sử dụng **Critical Section**
  - Truy nhập vào vùng tranh chấp:
    ```
    EnterCriticalSection(&cs);
    ```
    - Hàm tạm dừng luồng nếu luồng khác đang trong vùng tranh chấp.
    - Hàm trả về nếu không có luồng nào đang trong vùng tranh chấp.
  - Rời khỏi vùng tranh chấp:
    ```
    LeaveCriticalSection(&cs);
    ```

3.2 Đồng bộ và tránh xung đột trong lập trình đa luồng

- Sử dụng **Critical Section**

```
#define NUMTHREADS 4
CRITICAL_SECTION g_cs; // why does this have to be global?
int g_sum = 0;

DWORD WINAPI threadFunc(LPVOID arg)
{
    int mySum = bigComputation();
    EnterCriticalSection(&g_cs);
    g_sum += mySum;    // threads access one at a time
    LeaveCriticalSection(&g_cs);
    return 0;
}
main() {
    HANDLE hThread[NUMTHREADS];
    InitializeCriticalSection(&g_cs);
    for (int i = 0; i < NUMTHREADS; i++)
        hThread[i] = CreateThread(NULL, 0, threadFunc, NULL, 0, NULL);
    WaitForMultipleObjects(NUMTHREADS, hThread, TRUE, INFINITE);
    DeleteCriticalSection(&g_cs);
}
```

3.2 Đồng bộ và tránh xung đột trong lập trình đa luồng

Bài tập

```
static long num_steps = 100000;
double step, pi;

void main()
{
    int i;
    double x, sum = 0.0;

    step = 1.0 / (double)num_steps;
    for (i = 0; i < num_steps; i++) {
        x = (i + 0.5) * step;
        sum = sum + 4.0 / (1.0 + x * x);
    }
    pi = step * sum;
    printf("Pi = % f\n", pi);
}
```

Đoạn chương trình được sử dụng để tính xấp xỉ số PI.

Viết lại chương trình sử dụng multi-thread.

Chương 4. Các phương pháp vào ra trong lập trình socket

Chương 4. Các phương pháp vào ra

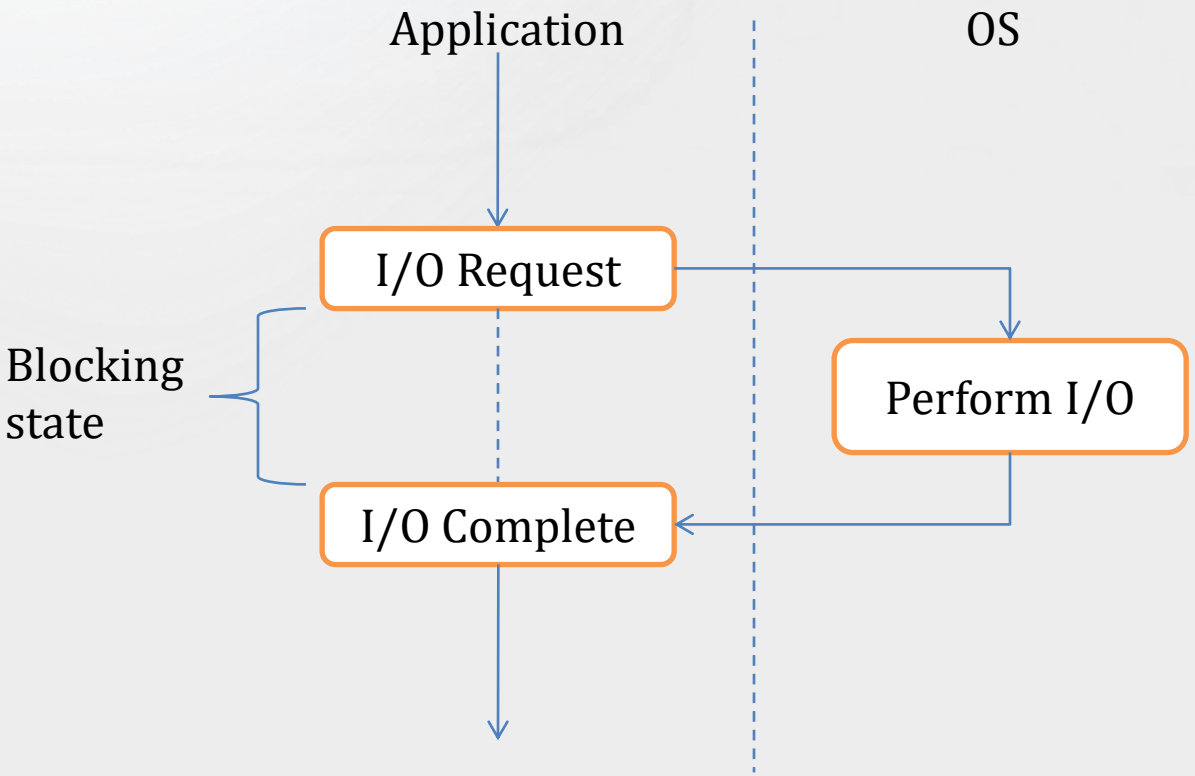
4.1. Các chế độ hoạt động của Winsock
4.2. Phương pháp vào ra sử dụng lập trình đa luồng
4.3. Phương pháp vào ra sử dụng hàm select
4.4. Phương pháp vào ra sử dụng hàm AsyncSelect
4.5. Phương pháp vào ra sử dụng hàm EventSelect
4.6. Phương pháp vào ra sử dụng cơ chế Overlapped
4.7. Phương pháp vào ra sử dụng cơ chế Overlapped – Completion Port

4.1 Các chế độ hoạt động của Winsock

- Blocking (Đồng bộ):
  - Là chế độ mà các hàm vào ra sẽ chặn thread đến khi thao tác vào ra hoàn tất (các hàm vào ra sẽ không trở về cho đến khi thao tác hoàn tất).
  - Là chế độ mặc định của SOCKET
  - Các hàm ảnh hưởng:
    - **accept**
    - **connect**
    - **send**
    - **recv**
    - **...**

4.1 Các chế độ hoạt động của Winsock

- Blocking (Đồng bộ):



4.1 Các chế độ hoạt động của Winsock

- Blocking (Đồng bộ):
  - Thích hợp với các ứng dụng xử lý tuần tự. Không nên gọi các hàm blocking khi ở thread xử lý giao diện (GUI Thread).
  - Ví dụ: Thread bị chặn bởi hàm **recv** thì không thể gửi dữ liệu

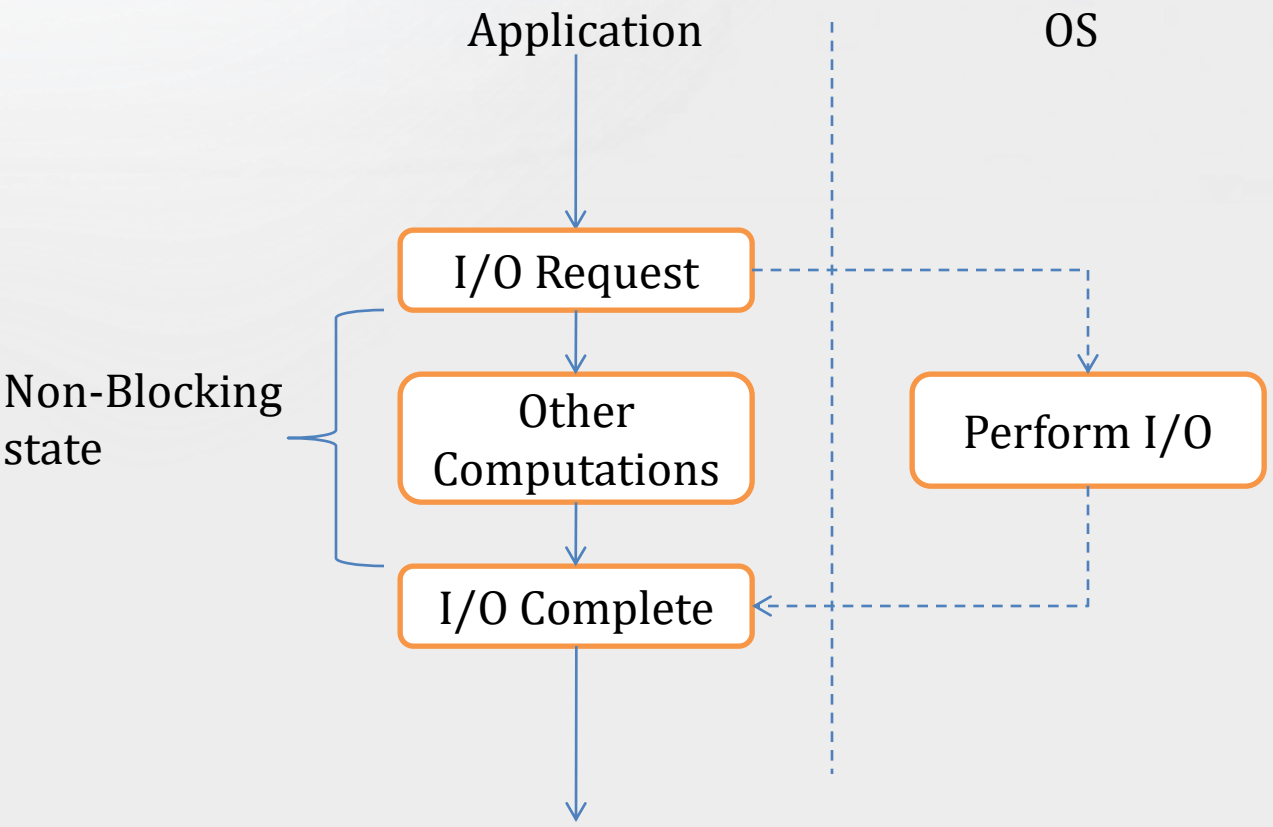
```
...
do {
    // Thread sẽ bị chặn lại khi gọi hàm recvfrom
    // Trong lúc đợi dữ liệu thì không thể gửi dữ liệu
    rc = recvfrom(receiver, szXau, 128, 0,
        (sockaddr*)&senderAddress, &senderLen);
    //...
} while ()
...
```

4.1 Các chế độ hoạt động của Winsock

- Non-Blocking (Bất đồng bộ):
  - Là chế độ mà các thao tác vào ra sẽ trở về nơi gọi ngay lập tức và tiếp tục thực thi thread. Kết quả của thao tác vào ra sẽ được thông báo cho chương trình dưới một cơ chế đồng bộ nào đó.
  - Các hàm vào ra bất đồng bộ sẽ trả về mã lỗi **WSAWOULDBLOCK** nếu thao tác đó không thể hoàn tất ngay và mất thời gian đáng kể (chấp nhận kết nối, nhận dữ liệu, gửi dữ liệu...). Đây là điều hoàn toàn bình thường.
  - Có thể sử dụng trong thread xử lý giao diện của ứng dụng.
  - Thích hợp với các ứng dụng hướng sự kiện.

4.1 Các chế độ hoạt động của Winsock

- Non-Blocking (Bất đồng bộ):



4.1 Các chế độ hoạt động của Winsock

- Non-Blocking (Bất đồng bộ):
  - Socket cần chuyển sang chế độ này bằng hàm **ioctlsocket**

```
SOCKET          s;
unsigned long ul = 1;
int             nRet;

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
// Chuyển sang chế độ non-blocking
nRet = ioctlsocket(s, FIONBIO, (unsigned long*)&ul);
if (nRet == SOCKET_ERROR) {
    // Thất bại
}
```

4.2 Vào ra sử dụng lập trình đa luồng

- Mô hình mặc định, đơn giản nhất.
- Không thể gửi nhận dữ liệu đồng thời trong cùng một luồng.
- Chỉ nên áp dụng trong các ứng dụng đơn giản, xử lý tuần tự, ít kết nối.
- Giải quyết vấn đề xử lý song song bằng việc tạo thêm các thread chuyên biệt: thread gửi dữ liệu, thread nhận dữ liệu
- Hàm API **CreateThread** được sử dụng để tạo một luồng mới

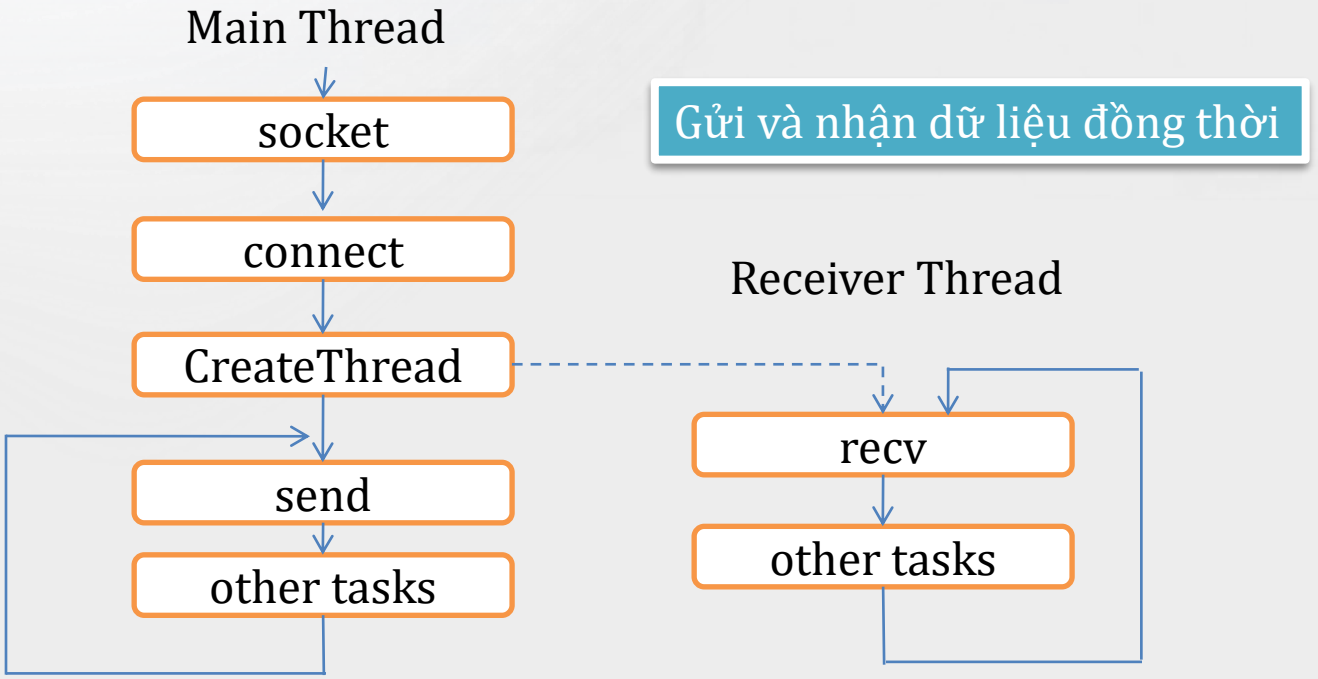
```
HANDLE WINAPI CreateThread(
    __in LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in SIZE_T dwStackSize,
    __in LPTHREAD_START_ROUTINE lpStartAddress,
    __in LPVOID lpParameter,
    __in DWORD dwCreationFlags,
    __out LPDWORD lpThreadId);
```

- Hàm API **TerminateThread** được sử dụng để xóa thread

```
BOOL WINAPI TerminateThread( __in_out HANDLE hThread,
                             __in DWORD dwExitCode );
```

4.2 Vào ra sử dụng lập trình đa luồng

- Ứng dụng client gửi nhận dữ liệu đồng thời



4.2 Vào ra sử dụng lập trình đa luồng

- Đoạn chương trình sau sẽ minh họa việc gửi và nhận dữ liệu đồng thời trong TCP Client

```
// Khai báo luồng xử lý việc nhận dữ liệu
DWORD WINAPI ReceiverThread(LPVOID lpParameter);
...
// Khai báo các biến toàn cục
SOCKADDR_IN address;
SOCKET client;
char szXau[128];
...
rc = connect(client, (sockaddr*)&address, sizeof(address));
// Tạo luồng xử lý việc nhận dữ liệu
CreateThread(0, 0, ReceiverThread, 0, 0, 0);
while (strlen(gets(szXau)) >= 2) {
    rc = send(client, szXau, strlen(szXau), 0);
}
...
```

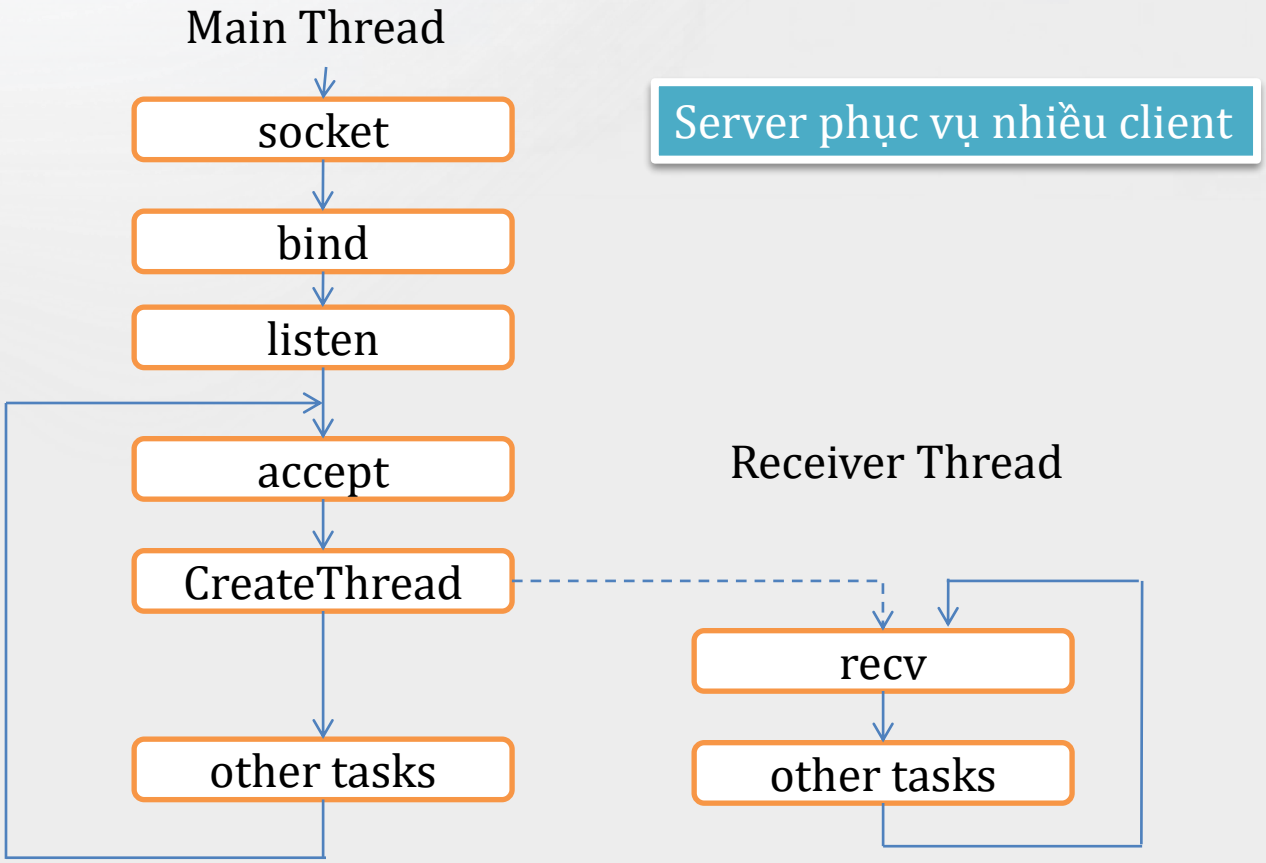
4.2 Vào ra sử dụng lập trình đa luồng

- Đoạn chương trình (tiếp)

```
DWORD WINAPI ReceiverThread(LPVOID lpParameter) {
    char szBuf[128];
    int len = 0;
    do {
        len = recv(client, szBuf, 128, 0);
        if (len >= 2) {
            szBuf[len] = 0;
            printf("%s\n", szBuf);
        }
        else
            break;
    } while (len >= 2);
}
```

4.2 Vào ra sử dụng lập trình đa luồng

- Ứng dụng server chấp nhận nhiều kết nối



Bài tập: Chat Server

Sử dụng mô hình đa luồng, viết chương trình chat server thực hiện các công việc sau:

- Nhận kết nối từ client, và vào vòng lặp hỏi tên client cho đến khi client gửi đúng cú pháp:

  "client_id: xxxxxxxx"

  trong đó xxxxxxx là tên
- Sau đó vào vòng lặp nhận dữ liệu từ một client và gửi dữ liệu đó đến các client còn lại, ví dụ: client có id "abc" gửi "xin chào" thì các client khác sẽ nhận được: "abc: xin chao" hoặc có thể thêm thời gian vào trước ví dụ: "2014/05/06 11:00:00PM abc: xin chao"

Bài tập: Telnet Server

Sử dụng mô hình đa luồng, viết chương trình telnet server làm nhiệm vụ sau:

- Khi đã kết nối với 1 client nào đó, yêu cầu client gửi user và pass, so sánh với file cơ sở dữ liệu là một file text, mỗi dòng chứa một cặp user + pass ví dụ:

  "admin admin
  guest nopass
  …"

  - Nếu so sánh sai thì báo lỗi đăng nhập
  - Nếu đúng thì đợi lệnh từ client, thực hiện lệnh và trả kết quả cho client
- Dùng hàm system("dir c:\temp > c:\\temp\\out.txt") để thực hiện lệnh
  - dir c:\temp là ví dụ lệnh dir mà client gửi
  - > c:\\temp\\out.txt để định hướng lại dữ liệu ra từ lệnh dir, khi đó kết quả lệnh dir sẽ được ghi vào file văn bản
- Chú ý: Nếu nhiều client kết nối thì file out.txt có thể bị xung đột truy nhập, do đó nên dùng EnterCriticalSection và LeaveCriticalSection để tránh xung đột

4.3 Vào ra sử dụng hàm select

- Là mô hình được sử dụng phổ biến.
- Sử dụng hàm **select** để thăm dò các sự kiện trên socket (gửi dữ liệu, nhận dữ liệu, kết nối thành công, yêu cầu kết nối ...).
- Hỗ trợ nhiều kết nối cùng một lúc.
- Có thể xử lý tập trung tất cả các socket trong cùng một thread (tối đa 1024).

4.3 Vào ra sử dụng hàm select

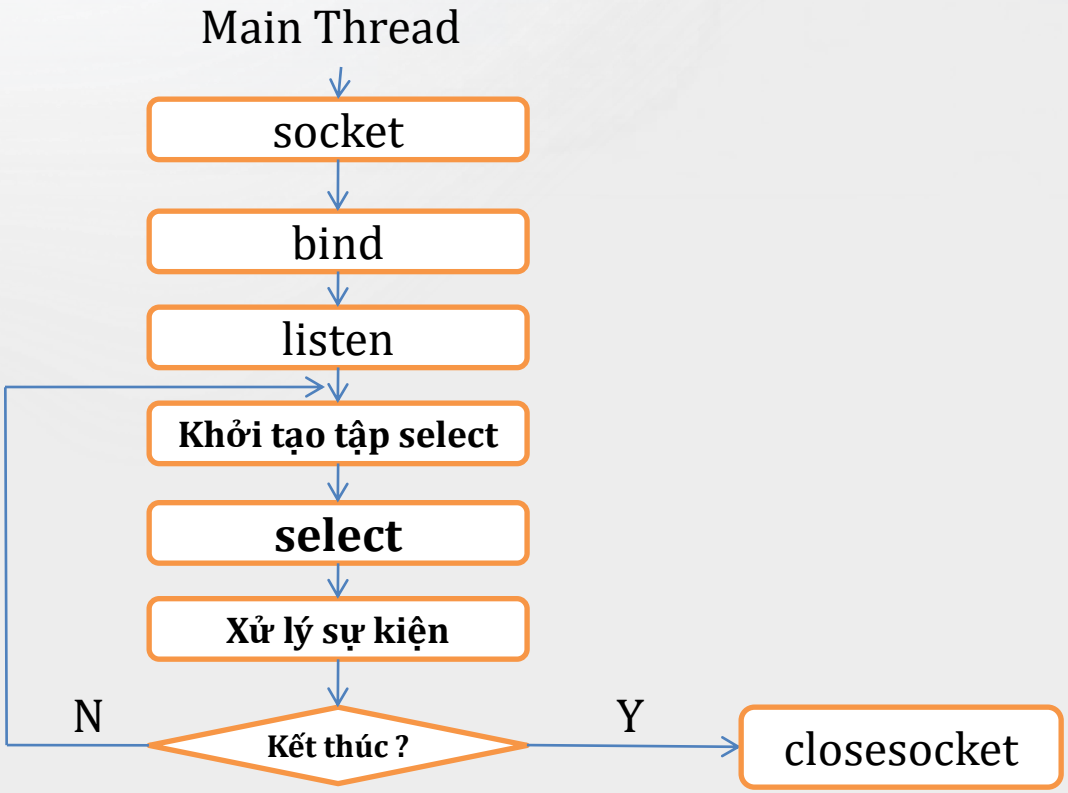
- Nguyên mẫu hàm như sau

```
int select(
    int nfds, // Không sử dụng
    fd_set FAR* readfds,   // Tập các socket hàm sẽ thăm dò cho sự kiện read
    fd_set FAR* writefds,  // Tập các socket hàm sẽ thăm dò cho sự kiện write
    fd_set FAR* exceptfds, // Tập các socket hàm sẽ thăm dò cho sự kiện except
    const struct timeval FAR* timeout // Thời gian thăm dò tối đa
);
```

- Giá trị trả về:
  - Thành công: số lượng socket có sự kiện xảy ra
  - Hết giờ: 0
  - Thất bại: SOCKET_ERROR

4.3 Vào ra sử dụng hàm select

- Mô hình **select**



4.3 Vào ra sử dụng hàm select

- Mô hình **select**
  - Điều kiện thành công của **select**
    - Một trong các socket của tập readfds nhận được dữ liệu hoặc kết nối bị đóng, reset, hủy, hoặc hàm accept thành công.
    - Một trong các socket của tập writefds có thể gửi dữ liệu, hoặc hàm connect thành công trên socket non-blocking.
    - Một trong các socket của tập exceptfds nhận được dữ liệu OOB, hoặc connect thất bại.
  - Các tập readfds, writefds, exceptfds có thể NULL, nhưng không thể cả ba cùng NULL.
  - Các MACRO FD_CLR, FD_ZERO, FD_ISSET, FD_SET sử dụng để thao tác với các cấu trúc fdset.

4.3 Vào ra sử dụng hàm select

- Mô hình **select**
  - Đoạn chương trình sau sẽ thăm dò trạng thái của socket s khi nào có dữ liệu

```
SOCKET  s;
fd_set  fdread;
int     ret;
// Khởi tạo socket s và tạo kết nối
// Thao tác vào ra trên socket s
while (TRUE) {
    // Xóa tập fdread
    FD_ZERO(&fdread);
    // Thêm s vào tập fdread
    FD_SET(s, &fdread);
    // Đợi sự kiện trên socket
    ret = select(0, &fdread, NULL, NULL, NULL);
    if (ret == SOCKET_ERROR) {
        // Xử lý lỗi
    }
```

4.3 Vào ra sử dụng hàm select

- Mô hình **select**
  - Đoạn chương trình (tiếp)

```
    if (ret > 0) {
        // Kiểm tra xem s có được thiết lập hay không
        if (FD_ISSET(s, &fdread)) {
            // Đọc dữ liệu từ s
        }
    }
}
```

4.3 Vào ra sử dụng hàm select

- Mô hình **select**
  - Ví dụ minh họa:
    - Client xử lý dữ liệu nhận được từ server
    - Server chấp nhận nhiều kết nối
    - Server chấp nhận nhiều kết nối và xử lý dữ liệu đối với các kết nối

Bài tập

- Viết lại chat server và telnet server với các yêu cầu như trước sử dụng cơ chế vào ra thăm dò (select)

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Cơ chế xử lý sự kiện dựa trên thông điệp của Windows
  - Ứng dụng GUI có thể nhận được các thông điệp từ WinSock qua cửa sổ của ứng dụng.
  - Hàm **WSAAsyncSelect** được sử dụng để chuyển socket sang chế độ bất đồng bộ và thiết lập tham số cho việc xử lý sự kiện

```
int WSAAsyncSelect(
    SOCKET s,//  [IN] Socket sẽ xử lý sự kiện
    HWND hWnd,//  [IN] Handle cửa sổ nhận sự kiện
    unsigned int wMsg, // [IN] Mã thông điệp, tùy chọn, thường
>= WM_USER
    long lEvent// [IN] Mặt nạ chứa các sự kiện ứng dụng muốn
nhận bao gồm FD_READ, FD_WRITE, FD_ACCEPT, FD_CONNECT, FD_CLOSE
);
```

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Ví dụ:

```
WSAAsyncSelect(s, hwnd, WM_SOCKET, FD_CONNECT | FD_READ |
               FD_WRITE | FD_CLOSE);
```

  - Tất cả các cửa sổ đều có hàm callback để nhận sự kiện từ Windows. Khi ứng dụng đã đăng ký socket với cửa sổ nào, thì cửa sổ đó sẽ nhận được các sự kiện của socket.
  - Nguyên mẫu của hàm callback của cửa số:

```
LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg,
                            WPARAM wParam, LPARAM lParam);
```

  - Khi cửa sổ nhận được các sự kiện liên quan đến WinSock:
    - uMsg sẽ chứa mã thông điệp mà ứng dụng đã đăng ký bằng WSAAsyncSelect
    - wParam chứa bản thân socket xảy ra sự kiện
    - Nửa cao của lParam chứa mã lỗi nếu có, nửa thấp chứa mã sự kiện có thể là FD_READ, FD_WRITE, FD_CONNECT, FD_ACCEPT, FD_CLOSE

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Ứng dụng sẽ dùng hai MACRO: WSAGETSELECTERROR và WSAGETSELECTEVENT để kiểm tra lỗi và sự kiện xảy ra trên socket.
  - Ví dụ:

```
BOOL CALLBACK WinProc(HWND hDlg, UINT wMsg, WPARAM wParam, LPARAM lParam) {
    SOCKET Accept;
    switch (wMsg) {
    case WM_PAINT:// Xử lý sự kiện khác
        break;
    case WM_SOCKET:  // Sự kiện WinSock
        if (WSAGETSELECTERROR(lParam)) // Kiểm tra lỗi hay không
        {
            closesocket((SOCKET)wParam); // Đóng socket
            break;
        }
```

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Ví dụ (tiếp):

```
            switch (WSAGETSELECTEVENT(lParam)) { // Xác định sự kiện
                case FD_ACCEPT:    // Chấp nhận kết nối
                    Accept = accept(wParam, NULL, NULL);
                    ...
                    break;
                case FD_READ:   // Có dữ liệu từ socket wParam
                    ...
                    break;
                case FD_WRITE:  // Có thể gửi dữ liệu đến socket wParam
                    break;
                case FD_CLOSE:   // Đóng kết nối
                    closesocket((SOCKET)wParam);
                    break;
            }
            break;
    }
    return TRUE;
}
```

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Tạo cửa sổ HWND: sử dụng hàm RegisterClass() và hàm CreateWindow()

```
WNDCLASS wndclass;
CHAR *providerClass = "AsyncSelect";
HWND window;

wndclass.style = 0;
wndclass.lpfnWndProc = (WNDPROC)WinProc;
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = NULL;
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = (LPCWSTR)providerClass;

if (RegisterClass(&wndclass) == 0)
        return NULL;
```

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Tạo cửa sổ HWND: sử dụng hàm RegisterClass() và hàm CreateWindow()

```
// Create a window
if ((window = CreateWindow((LPCWSTR)providerClass, L"", WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
          NULL, NULL, NULL, NULL)) == NULL)
return NULL;
```

  - Vòng lặp để truyền và nhận các thông điệp cửa sổ trong hàm main()

```
MSG msg;
while (GetMessage(&msg, NULL, 0, 0) > 0)
{
          TranslateMessage(&msg);
          DispatchMessage(&msg);
}
```

4.4 Vào ra sử dụng hàm AsyncSelect

- Mô hình **WSAAsyncSelect**
  - Ưu điểm: xử lý hiệu quả nhiều sự kiện trong cùng một luồng.
  - Nhược điểm: ứng dụng phải có ít nhất một cửa sổ, không nên dồn quá nhiều socket vào cùng một cửa sổ vì sẽ dẫn tới đình trệ trong việc xử lý giao diện.

4.4 Vào ra sử dụng hàm AsyncSelect

- **Ví dụ**
  - Ứng dụng client (console).
  - Ứng dụng server (console).
  - Ứng dụng server (GUI).

Bài tập

- Viết lại chat server và telnet server với các yêu cầu như trước sử dụng cơ chế không đồng bộ bằng WSAAsyncSelect.
- Viết ứng dụng chat client sử dụng giao diện cửa sổ.

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**
  - Xử lý dựa trên cơ chế đồng bộ đối tượng sự kiện của Windows: **WSAEVENT**
  - Mỗi đối tượng có hai trạng thái: Báo hiệu (signaled) và chưa báo hiệu (non-signaled).
  - Hàm **WSACreateEvent** sẽ tạo một đối tượng sự kiện ở trạng thái chưa báo hiệu và có chế độ hoạt động là thiết lập thủ công (manual reset).

    ```
    WSAEVENT WSACreateEvent(void);
    ```

  - Hàm **WSAResetEvent** sẽ chuyển đối tượng sự kiện về trạng thái chưa báo hiệu

    ```
    BOOL WSAResetEvent(WSAEVENT hEvent);
    ```

  - Hàm **WSACloseEvent** sẽ giải phóng một đối tượng sự kiện

    ```
    BOOL WSACloseEvent(WSAEVENT hEvent);
    ```

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**
  - Hàm **WSAEventSelect** sẽ tự động chuyển socket sang chế độ non-blocking và gắn các sự kiện của socket với đối tượng sự kiện truyền vào theo tham số

```
int WSAEventSelect(
    SOCKET s, // [IN] Socket cần xử lý sự kiện
    WSAEVENT hEventObject, // [IN] Đối tượng sự kiện
                           // đã tạo trước đó
    long lNetworkEvents // [IN] Các sự kiện ứng dụng
                        // muốn nhận từ  WinSock
);
```

  - Ví dụ:

```
rc = WSAEventSelect(s, hEventObject, FD_READ |
FD_CLOSE);
```

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**
  - Hàm **WaitForMultipleEvent** sẽ đợi sự kiện trên một mảng các đối tượng sự kiện cho đến khi một trong các đối tượng chuyển sang trạng thái báo hiệu.

```
DWORD WSAWaitForMultipleEvents(
  DWORD cEvents,   //[IN] Số lượng sự kiện cần đợi
  const WSAEVENT FAR * lphEvents, //[IN] Mảng sự kiện, max 64
  BOOL fWaitAll,   //[IN] Có đợi tất cả các sự kiện không ?
  DWORD dwTimeout, //[IN] Thời gian đợi tối đa
  BOOL fAlertable  //[IN] Thiết lập là FALSE
);
```

  Giá trị trả về
  - Thành công: Số thứ tự của sự kiện xảy ra + **WSA_WAIT_EVENT_0**.
  - Hết giờ: WSA_WAIT_TIMEOUT.
  - Thất bại: WSA_WAIT_FAILED.

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**

Xử lý nhiều sự kiện:

```
WSAEVENT events[WSA_MAXIMUM_WAIT_EVENTS];
int count = 0, ret, index;

// Assign event handles into events
while (1) {
    ret = WSAWaitForMultipleEvents(count, events, FALSE, WSA_INFINITE, FALSE);
    if ((ret != WSA_WAIT_FAILED) && (ret != WSA_WAIT_TIMEOUT)) {
        index = ret - WSA_WAIT_EVENTT_0;
        // Service event signaled on events[index]
        WSAResetEvent(events[index]);
    }
}
```

Vấn đề: nếu sự kiện đầu tiên luôn xảy ra, các sự kiện khác có thể bị bỏ qua
Cách xử lý: kiểm tra từng sự kiện còn lại khi 1 trong các sự kiện xảy ra

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**

Xử lý nhiều sự kiện:

```
WSAEVENT events[WSA_MAXIMUM_WAIT_EVENTS];
int count = 0, ret, index;

// Assign event handles into events
while (1) {
    ret = WSAWaitForMultipleEvents(count, events, FALSE, WSA_INFINITE, FALSE);
    index = ret - WSA_WAIT_OBJECT_0;
    for (i = index; i < count; i++) {
        ret = WSAWaitForMultipleEvents(1, &events[i], TRUE, 0, FALSE);
        if ((ret != WSA_WAIT_FAILED) && (ret != WSA_WAIT_TIMEOUT))
        {
            // Service event signaled on events[index]
        }
    }
}
```

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**
  - Xác định mã của sự kiện gắn với một đối tượng sự kiện cụ thể bằng hàm **WSAEnumNetworkEvents.**

```
int WSAEnumNetworkEvents (
    SOCKET s, // [IN] Socket muốn thăm dò
    WSAEVENT hEventObject, // [IN] Đối tượng sự kiện tương ứng
    LPWSANETWORKEVENTS lpNetworkEvents // [OUT] Cấu trúc chứa mã sự kiện
);
```

  - Mã sự kiện lại nằm trong cấu trúc WSANETWORKEVENTS có khai báo như sau

```
typedef struct _WSANETWORKEVENTS {
   long lNetworkEvents; // Mặt nạ chứa sự kiện được kích hoạt
   int  iErrorCode[FD_MAX_EVENTS]; // Mảng các mã sự kiện
} WSANETWORKEVENTS, FAR * LPWSANETWORKEVENTS;
```

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**

```
if (NetworkEvents.lNetworkEvents & FD_ACCEPT) {
    if (NetworkEvents.iErrorCode[FD_ACCEPT_BIT] != 0) {
        printf("FD_ACCEPT failed with error %d\n",
NetworkEvents.iErrorCode[FD_ACCEPT_BIT]);
    }

    // Process ACCEPT event
    // Accept connection
}
if (NetworkEvents.lNetworkEvents & FD_READ) {
    if (NetworkEvents.iErrorCode[FD_READ_BIT] != 0) {
        printf("FD_READ failed with error %d\n",
NetworkEvents.iErrorCode[FD_READ_BIT]);
    }

    // Process READ event
    // Read data from socket
}
```

4.5 Vào ra sử dụng hàm EventSelect

- Mô hình **WSAEventSelect**

Ví dụ lập trình server sử dụng mô hình WSAEventSelect

Bài tập

- Viết lại chat server và telnet server với các yêu cầu như trước sử dụng cơ chế không đồng bộ bằng WSAEventSelect

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped**
  - Sử dụng cấu trúc OVERLAPPED chứa thông tin về thao tác vào ra.
  - Các thao tác vào ra sẽ trở về ngay lập tức và thông báo lại cho ứng dụng theo một trong hai cách sau:
    - **Event** được chỉ ra trong cấu trúc OVERLAPPED.
    - **Completion routine** được chỉ ra trong tham số của lời gọi vào ra.
  - Các hàm vào ra sử dụng mô hình này:
    - WSASend
    - WSASendTo
    - **WSARecv**
    - WSARecvFrom
    - WSAIoctl
    - WSARecvMsg
    - AcceptEx
    - ConnectEx
    - TransmitFile
    - TransmitPackets
    - DisconnectEx
    - WSANSPIoctl

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua **Event**

  Cấu trúc OVERLAPPED

```
typedef struct WSAOVERLAPPED
{
    DWORD    Internal;
    DWORD    InternalHigh;
    DWORD    Offset;
    DWORD    OffsetHigh;
    WSAEVENT hEvent;
} WSAOVERLAPPED, FAR * LPWSAOVERLAPPED
```

  - **Internal, InternalHigh, Offset, OffsetHigh** được sử dụng nội bộ trong WinSock
  - **hEvent** là đối tượng **WSAEVENT** sẽ được báo hiệu khi thao tác vào ra hoàn tất, chương trình cần khởi tạo cấu trúc với một đối tượng sự kiện hợp lệ.
  - Khi thao tác vào ra hoàn tất, chương trình cần lấy kết quả vào ra thông qua hàm **WSAGetOverlappedResult**

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua **Event**

  Hàm **WSARecv**

```
int WSARecv(
  SOCKET s,
  LPWSABUF lpBuffers,
  DWORD dwBufferCount,
  LPDWORD lpNumberOfBytesRecvd,
  LPDWORD lpFlags,
  LPWSAOVERLAPPED lpOverlapped,
  LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

  - **s** là socket nhận dữ liệu
  - **lpBuffers** là con trỏ đến cấu trúc WSABUF
  - **dwBufferCount** số lượng cấu trúc buffer trong mảng lpBuffers
  - **lpNumberOfBytesRecvd** là con trỏ chỉ ra số byte nhận được
  - **lpFlags** là con trỏ đến các cờ sử dụng để thay đổi thao tác lệnh WSARecv
  - **lpOverlapped** là con trỏ đến cấu trúc Overlapped
  - **lpCompletionRoutine** không sử dụng
  - Hàm trả về 0 nếu không có lỗi, SOCKET_ERROR nếu có lỗi

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua **Event**

  Cấu trúc WSABUF

```
typedef struct WSABUF
{
    u_long    len;
    char FAR  *buf;
} WSABUF, *LPWSABUF
```

  - **len:** độ dài của buffer
  - **buf:** con trỏ buffer

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua **Event**

  Hàm **WSAGetOverlappedResult**

```
BOOL WSAGetOverlappedResult(
    SOCKET s,
    LPWSAOVERLAPPED lpOverlapped,
    LPDWORD lpcbTransfer,
    BOOL fWait,
    LPDWORD lpdwFlags
);
```

  - **s** là socket muốn kiểm tra kết quả
  - **lpOverlapped** là con trỏ đến cấu trúc OVERLAPPED
  - **lpcbTransfer** là con trỏ đến biến sẽ lưu số byte trao đổi được
  - **fWait** là biến báo cho hàm đợi cho đến khi thao tác vào ra hoàn tất
  - **lpdwFlags** : cờ kết quả của thao tác
  - Hàm trả về TRUE nếu thao tác hoàn tất hoặc FALSE nếu thao tác chưa hoàn tất, có lỗi hoặc không thể xác định.

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua **event**
  - Tạo đối tượng event với **WSACreateEvent**.
  - Khởi tạo cấu trúc OVERLAPPED với event vừa tạo.
  - Gửi yêu cầu vào ra với tham số là cấu trúc OVERLAPPED vừa tạo, **tham số liên quan đến CompletionRoutine phải luôn bằng NULL**.
  - Đợi thao tác kết thúc qua hàm **WSAWaitForMultipleEvents.**
  - Nhận kết quả vào ra qua hàm **WSAGetOverlappedResult**

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Ví dụ xử lý qua **event**


```
// Khoi tao server va chap nhan ket noi

OVERLAPPED overlapped;
WSAEVENT receivedEvent = WSACreateEvent();// Tao doi tuong event
memset(&overlapped, 0, sizeof(overlapped));// Khoi tao cau truc overlapped
overlapped.hEvent = receivedEvent;// Gan event voi cau truc overlapped

char buf[256];
WSABUF databuf;// Khai bao cau truc buffer nhan du lieu
databuf.buf = buf;
databuf.len = sizeof(buf);

DWORD bytesReceived;
DWORD flags = 0;
int ret;

while (1)
{
    // Gui yeu cau nhan du lieu
    ret = WSARecv(client, &databuf, 1, &bytesReceived, &flags, &overlapped, FALSE);
```

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Ví dụ xử lý qua **event**

```
if (ret == SOCKET_ERROR)
{
     ret = WSAGetLastError();
     // Neu loi la WSA_IO_PENDING nghia la dang cho du lieu
     if (ret != WSA_IO_PENDING)
     {
          printf("WSARecv() failed: error code %d\n", ret);
          break;
     }
}

// Doi den khi co du lieu duoc gui den
ret = WSAWaitForMultipleEvents(1, &receivedEvent, FALSE, 5000, FALSE);
if (ret == WSA_WAIT_FAILED)
{
     printf("WSAWaitForMultipleEvents() failed\n");
     break;
}
```

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Ví dụ xử lý qua **event**

```
    if (ret == WSA_WAIT_TIMEOUT)
    {
        printf("Timed out\n");
        continue;
    }

    WSAResetEvent(receivedEvent);

    // Nhan du lieu vao buffer
    ret = WSAGetOverlappedResult(client, &overlapped, &bytesReceived, FALSE, &flags);
    if (bytesReceived == 0)
        break;

    // Xu ly du lieu
    buf[bytesReceived] = 0;
    printf("Received: %s\n", buf);
}
```

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua Completion Routine
  - Hệ thống sẽ thông báo cho ứng dụng biết thao tác vào ra kết thúc thông qua một hàm callback gọi là **Completion Routine**
  - Nguyên mẫu của hàm như sau

```
void CALLBACK CompletionROUTINE(
   IN DWORD dwError,   // Mã lỗi
   IN DWORD dwTransferred,    // Số byte trao đổi
   IN LPWSAOVERLAPPED lpOverlapped,  // Cấu trúc
overlapped tương ứng
   IN DWORD dwFlags ); // Cờ kết quả thao tác vào ra
```

  - WinSock sẽ bỏ qua trường **event** trong cấu trúc OVERLAPPED, việc tạo đối tượng event và thăm dò là không cần thiết nữa.

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Xử lý qua Completion Routine
  - Ứng dụng cần chuyển luồng sang trạng thái **alertable** ngay sau khi gửi yêu cầu vào ra.
  - Các hàm có thể chuyển luồng sang trạng thái **alertable**: **`WSAWaitForMultipleEvents, SleepEx`**
  - Nếu ứng dụng không có đối tượng event nào thì có thể sử dụng SleepEx

```
DWORD SleepEx(
   DWORD dwMilliseconds, // Thời gian đợi
   BOOL  bAlertable  );  //  Trạng thái alertable
```

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Ví dụ Completion Routine


```
// Khai bao bien toan cuc
SOCKET client;
char buf[256];
WSABUF databuf;

// Protype cua ham callback
void CALLBACK CompletionRoutine(DWORD, DWORD, LPWSAOVERLAPPED, DWORD);

int main()
{
    // Chap nhan ket noi va truyen nhan du lieu

    OVERLAPPED overlapped;
    memset(&overlapped, 0, sizeof(overlapped));

    databuf.buf = buf;
    databuf.len = sizeof(buf);

    DWORD bytesReceived;
    DWORD flags = 0;
```

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Ví dụ Completion Routine

```
    // Gui yeu cau du lieu lan dau
    int ret = WSARecv(client, &databuf, 1, &bytesReceived, &flags,
&overlapped, CompletionRoutine);

    // Chuyen luong sang trang thai alertable
    while (1)
    {
        ret = SleepEx(5000, TRUE);
        if (ret == 0)
        {
            printf("Timed out\n");
        }
    }

    closesocket(client);
    closesocket(listener);
    WSACleanup();
}
```

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Overlapped** – Ví dụ Completion Routine

```
void CALLBACK CompletionRoutine(DWORD dwError, DWORD dwBytesReceived,
LPWSAOVERLAPPED lpOverlapped, DWORD dwFlags)
{
    // Kiem tra loi
    if (dwError != 0 || dwBytesReceived == 0)
    {
        closesocket(client);
    return;
    }

    // Xu ly du lieu nhan duoc
    buf[dwBytesReceived] = 0;
    printf("Received: %s\n", buf);

    // Gui yeu cau du lieu tiep theo
    dwFlags = 0;
    int ret = WSARecv(client, &databuf, 1, &dwBytesReceived, &dwFlags,
lpOverlapped, CompletionRoutine);
}
```

4.6 Vào ra sử dụng cơ chế Overlapped

- **Bài tập:** Sử dụng kỹ thuật lập trình đa luồng để tạo server chấp nhận nhiều kết nối và nhận dữ liệu cho mỗi kết nối thông qua cơ chế Overlapped
  - Sử dụng đối tượng Event
  - Sử dụng Completion Routine

4.6 Vào ra sử dụng cơ chế Overlapped

- Mô hình **Completion Port**
  - Có hiệu năng tốt nhất khi so sánh với các mô hình khác trong việc quản lý nhiều kết nối
  - Cơ chế khởi tạo phức tạp hơn so với các mô hình khác
  - Đối tượng Completion Port được tạo ra để quản lý các yêu cầu vào ra Overlapped IO (WSARecv, WSASend, …)
  - Sử dụng các thread để phục vụ khi các yêu cầu vào ra hoàn tất

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Các bước khởi tạo cơ bản
    1. Tạo đối tượng CompletionPort sử dụng hàm CreateIoCompletionPort()
    2. Xác định số processor của hệ thống
    3. Tạo các worker thread để phục vụ các yêu cầu vào ra, số lượng thread tương ứng với số lượng processor
    4. Tạo đối tượng socker chờ các kết nối
    5. Chấp nhận kết nối mới
    6. Tạo cấu trúc dữ liệu cho kết nối
    7. Gắn kết nối với đối tượng completion port, sử dụng hàm CreateIoCompletionPort()
    8. Gửi yêu cầu dữ liệu lần đầu bằng lệnh WSARecv()
    9. Lặp lại các bước từ 5 đến 8 cho đến khi server ngừng hoạt động

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Hàm **CreateIoCompletionPort()** được sử dụng để tạo đối tượng CompletionPort

```
HANDLE CreateIoCompletionPort(
    HANDLE FileHandle,
    HANDLE ExistingCompletionPort,
    DWORD CompletionKey,
    DWORD NumberOfConcurrentThreads
);
```

  - Hàm này có 2 chức năng chính:
    - Tạo mới đối tượng CompletionPort
    - Gắn một handle với đối tượng CompletionPort
  - Khi tạo mới đối tượng CompletionPort: tham số `NumberOfConcurrentThreads` định nghĩa số luồng chạy đồng thời trên một đối tượng CompletionPort, nên thiết lập bằng với số processor của hệ thống (tham số tương ứng bằng 0)

```
CompletionPort = CreateIoCompletionPort(INVALID_HANDLE_VALUE,
NULL, 0, 0);
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Số processor của hệ thống có thể được xác định thông qua hàm API **GetSystemInfo()**

```
SYSTEM_INFO SystemInfo;
// Xác định số lượng processor của hệ thống
GetSystemInfo(&SystemInfo);

// Tạo các thread tương ứng với số lượng processor
for(i = 0; i < SystemInfo.dwNumberOfProcessors; i++)
{
    // Đối tượng CompletionPort được truyền vào các thread

    HANDLE ThreadHandle = CreateThread(NULL, 0,
        ServerWorkerThread, CompletionPort, 0, NULL);
    CloseHandle(ThreadHandle);
}
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Hàm **CreateIoCompletionPort()** được sử dụng để gắn mỗi client socket với đối tượng CompletionPort

```
HANDLE CreateIoCompletionPort(
    HANDLE FileHandle,
    HANDLE ExistingCompletionPort,
    DWORD CompletionKey,
    DWORD NumberOfConcurrentThreads
);
```

  - Các tham số cần quan tâm
    - **FileHandle**: handle của client socket
    - **ExistingCompletionPort**: đối tượng CompletionPort
    - **CompletionKey**: xác định cấu trúc dữ liệu gắn với socket (Per-Handle data)

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Cấu trúc dữ liệu PER_HANDLE_DATA

```
typedef struct _PER_HANDLE_DATA
{
   SOCKET Socket;                  // Lưu socket handle
   SOCKADDR_STORAGE ClientAddr; // Lưu địa chỉ client
} PER_HANDLE_DATA, * LPPER_HANDLE_DATA;
```

  - Ví dụ:

```
CreateIoCompletionPort((HANDLE)Accept, CompletionPort,
      (ULONG_PTR)PerHandleData, 0);
```

  - Cấu trúc dữ liệu được sử dụng trong các thread để lấy thông tin của client socket

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Trong các thread, hàm **GetQueueCompletionStatus()** được sử dụng để chờ đến khi yêu cầu vào ra hoàn tất

```
BOOL GetQueuedCompletionStatus(
    HANDLE CompletionPort,
    LPDWORD lpNumberOfBytesTransferred,
    PULONG_PTR lpCompletionKey,
    LPOVERLAPPED * lpOverlapped,
    DWORD dwMilliseconds
);
```

  - Các tham số:
    - **CompletionPort**: đối tượng **CompletionPort** được truyền vào thread
    - **lpNumberOfBytesTransferred:** con trỏ trả về số byte truyền nhận được
    - **lpCompletionKey:** con trỏ trả về cấu trúc dữ liệu của **socket**
    - **lpOverlapped:** con trỏ trả về cấu trúc dữ liệu của đối tượng **Overlapped**
    - **dwMilliseconds:** thời gian chờ, truyền vào giá trị **INFINITE** nếu chờ vô hạn

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port**
  - Cấu trúc dữ liệu của đối tượng Overlapped **PER_IO_DATA**

```
typedef struct _PER_IO_DATA
{
   OVERLAPPED Overlapped;
   WSABUF DataBuf;
   char buf[1024];
} PER_IO_DATA, * LPPER_IO_DATA;
```

  - Cấu trúc được sử dụng trong các thread để lấy dữ liệu trả về từ các thao tác vào ra
  - Ví dụ:

```
bool ret = GetQueuedCompletionStatus(CompletionPort, &BytesTransferred,
(PULONG_PTR)&PerHandleData, (LPOVERLAPPED *)&PerIoData, INFINITE);
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ


```
#include <stdio.h>
#include <winsock2.h>

#pragma comment(lib, "ws2_32")

// Khai bao cau truc du lieu socket
typedef struct _PER_HANDLE_DATA
{
    SOCKET Socket;
    SOCKADDR_STORAGE ClientAddr;
} PER_HANDLE_DATA, * LPPER_HANDLE_DATA;

// Khai bao cau truc du lieu overlapped
typedef struct _PER_IO_DATA
{
    OVERLAPPED Overlapped;
    WSABUF DataBuf;
    char buf[1024];
} PER_IO_DATA, * LPPER_IO_DATA;

DWORD WINAPI ServerWorkerThread(LPVOID);
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ

```
int main()
{
    WSADATA wsa;
    WSAStartup(MAKEWORD(2, 2), &wsa);

    // Tao doi tuong CompletionPort
    HANDLE completionPort = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL,
0, 0);

    // Lay thong tin he thong
    SYSTEM_INFO systemInfo;
    GetSystemInfo(&systemInfo);

    // Tao cac worker thread ung voi so processor
    for (int i = 0; i < systemInfo.dwNumberOfProcessors; i++)
    {
        HANDLE hThread = CreateThread(NULL, 0, ServerWorkerThread,
completionPort, 0, NULL);
        CloseHandle(hThread);
    }
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ

```
    SOCKET listener = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(9000);

    bind(listener, (SOCKADDR*)&addr, sizeof(addr));
    listen(listener, 5);

    while (1)
    {
        SOCKADDR_IN clientAddr;
        int clientAddrLen = sizeof(clientAddr);

        // Chap nhan ket noi
        SOCKET client = accept(listener, (SOCKADDR*)&clientAddr,
&clientAddrLen);
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ

```
        // Cap phat bo nho cho cau truc du lieu
        PER_HANDLE_DATA* pHandle = (LPPER_HANDLE_DATA)GlobalAlloc(GPTR,
sizeof(PER_HANDLE_DATA));

        // Luu thong tin socket vao cau truc du lieu socket
        printf("Socket number %d connected\n", client);
        pHandle->Socket = client;
        memcpy(&pHandle->ClientAddr, &clientAddr, clientAddrLen);

        // Gan socket voi doi tuong CompletionPort
        CreateIoCompletionPort((HANDLE)client, completionPort,
(ULONG_PTR)pHandle, 0);
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ

```
        // Cap phat bo nho cho cau truc du lieu vao ra overlapped
        PER_IO_DATA* pIO = (LPPER_IO_DATA)GlobalAlloc(GPTR, sizeof(PER_IO_DATA));

        pIO->DataBuf.len = sizeof(pIO->buf);
        pIO->DataBuf.buf = pIO->buf;

        DWORD bytesReceived, flags = 0;
        // Gui yeu cau du lieu lan dau
        WSARecv(client, &(pIO->DataBuf), 1, &bytesReceived, &flags, &(pIO-
>Overlapped), NULL);
    }

    return 0;
}
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ

```
// Worker thread xu ly viec cho du lieu va xu ly du lieu
DWORD WINAPI ServerWorkerThread(LPVOID lpParam)
{
    HANDLE completionPort = (HANDLE)lpParam;
    LPPER_HANDLE_DATA pHandle;
    LPPER_IO_DATA pIO;
    DWORD bytesReceived;
    DWORD flags = 0;

    while (TRUE)
    { // Cho den khi 1 yeu cau vao ra hoan tat
        bool ret = GetQueuedCompletionStatus(completionPort, &bytesReceived,
(PULONG_PTR)&pHandle, (LPOVERLAPPED*)&pIO, INFINITE);
        if (bytesReceived == 0) // Neu ket noi bi ngat
        {
            closesocket(pHandle->Socket);// Ngat ket noi
            GlobalFree(pHandle); // Giai phong bo nho da cap phat
            GlobalFree(pIO);
            continue;
        }
```

4.7 Vào ra sử dụng cơ chế Overlapped - Completion Port

- Mô hình **Completion Port** – Ví dụ

```
        // Xu ly du lieu nhan duoc
        pIO->buf[bytesReceived] = 0;
        printf("Received Data: %s\n", pIO->buf);

        // Gui yeu cau du lieu tiep theo
        WSARecv(pHandle->Socket, &(pIO->DataBuf), 1, &bytesReceived, &flags,
&(pIO->Overlapped), NULL);
    }
}
```

So sánh hiệu năng của các mô hình

Table 6-3 *I/O Method Performance Comparison*

I/O Model	Attempted/Connected	Memory Used (KB)	Non-Paged Pool	CPU Usage	Threads	Throughput (Send/ Receive Bytes Per Second)
Blocking	7000/ 1008	25,632	36,121	10–60%	2016	2,198,148/ 2,198,148
	12,000/ 1008	25,408	36,352	5– 40%	2016	404,227/ 402,227
Non- blocking	7000/ 4011	4208	135,123	95–100%*	1	0/0
	12,000/ 5779	5224	156,260	95–100%*	1	0/0
WSA- Async Select	7000/ 1956	3640	38,246	75–85%	3	1,610,204/ 1,637,819
	12,000/ 4077	4884	42,992	90–100%	3	652,902/ 652,902
WSA- Event Select	7000/ 6999	10,502	36,402	65–85%	113	4,921,350/ 5,186,297
	12,000/ 11,080	19,214	39,040	50–60%	192	3,217,493/ 3,217,493
	46,000/ 45,933	37,392	121,624	80–90%	791	3,851,059/ 3,851,059
Over- lapped (events)	7000/ 5558	21,844	34,944	65–85%	66	5,024,723/ 4,095,644
	12,000/12,000	60,576	48,060	35–45%	195	1,803,878/ 1,803,878
	49,000/48,997	241,208	155,480	85–95%	792	3,865,152/ 3,834,511
Over- lapped (comple- tion port)	7000/ 7000	36,160	31,128	40–50%	2	6,282,473/ 3,893,507
	12,000/12,000	59,256	38,862	40–50%	2	5,027,914/ 5,027,095
	50,000/49,997	242,272	148,192	55–65%	2	4,326,946/ 4,326,496

Bài tập: Chat server

Bài tập: Viết chương trình chat server phục vụ các client làm việc sau:
Nhận kết nối từ client, vào vòng lặp hỏi tên client cho đến khi client gửi đúng cú pháp:

CONNECT client_id

trong đó **client_id** là chuỗi ký tự không chứa dấu cách

Sau đó vào vòng lặp nhận và thực hiện các lệnh từ client:

LIST – liệt kê id của tất cả các client đã đăng nhập

SEND client_id message – gửi tin nhắn đến client có id là **client_id,** tin nhắn có định dạng **sender_client_id message**, nếu gửi thành công thì phản hồi lại cho client gửi là **OK**, nếu không thì phản hồi là **ERROR**

SEND ALL message – gửi tin nhắn đến tất cả các client đã đăng nhập, nếu thành công thì phản hồi **OK**, nếu không thì phản hồi **ERROR**

DISCONNECT – thoát khỏi trạng thái đăng nhập

Chương 5. Tìm hiểu và cài đặt một số giao thức phổ biến

Chương 5. Tìm hiểu và cài đặt một số giao thức phổ biến

5.1. Giao thức HTTP
5.2. Giao thức FTP
5.3. Tìm hiểu giao thức POP3

5.1 Giao thức HTTP

5.1.1. Tìm hiểu về giao thức HTTP
5.1.2. Một số vấn đề xử lý HTTP request
5.1.2. Lập trình ứng dụng máy chủ HTTP file

5.1.1 Tìm hiểu về giao thức HTTP

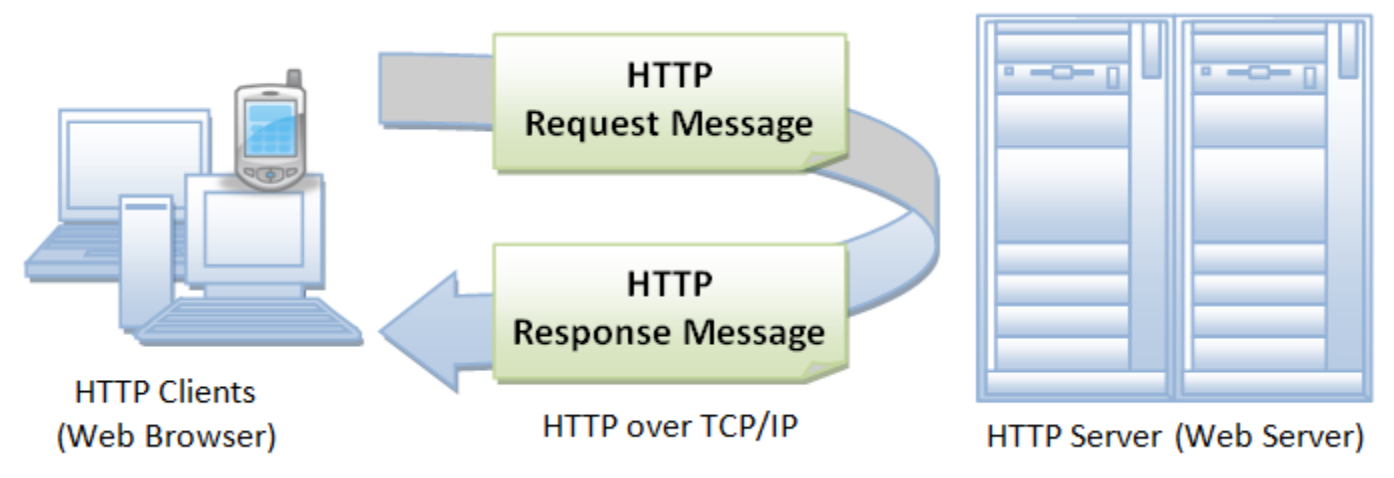
Định nghĩa:

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol - Giao thức truyền tải siêu văn bản) là một trong các giao thức chuẩn về mạng Internet, được dùng để liên hệ thông tin giữa Máy cung cấp dịch vụ (Web server) và Máy sử dụng dịch vụ (Web client), là giao thức Client/Server dùng cho World Wide Web – WWW.

HTTP là một giao thức ứng dụng của bộ giao thức TCP/IP (các giao thức nền tảng cho Internet).

5.1.1 Tìm hiểu về giao thức HTTP

Sơ đồ hoạt động:



- HTTP hoạt động dựa trên mô hình Client – Server. Trong mô hình này, các máy tính của người dùng sẽ đóng vai trò làm máy khách (Client). Sau một thao tác nào đó của người dùng, các máy khách sẽ gửi yêu cầu đến máy chủ (Server) và chờ đợi câu trả lời từ những máy chủ này.
- HTTP là một **stateless protocol**. Hay nói cách khác, request hiện tại không biết những gì đã hoàn thành trong request trước đó.

5.1.1 Tìm hiểu về giao thức HTTP

HTTP Requests:

Là phương thức để chỉ ra hành động mong muốn được thực hiện trên tài nguyên đã xác định.

Cấu trúc của một HTTP Request:

- **Request-line = Phương thức + URI–Request + Phiên bản HTTP .** Giao thức HTTP định nghĩa một tập các phương thức GET, POST, HEAD, PUT ... Client có thể sử dụng một trong các phương thức đó để gửi request lên server.
- Có thể có hoặc không các trường **header**: Các trường header cho phép client truyền thông tin bổ sung về yêu cầu, và về chính client, đến server. Một số trường: Accept-Charset, Accept-Encoding, Accept-Language, Authorization, Expect, From, Host, …
- Một dòng trống để đánh dấu sự kết thúc của các trường Header.
- Tùy chọn một thông điệp

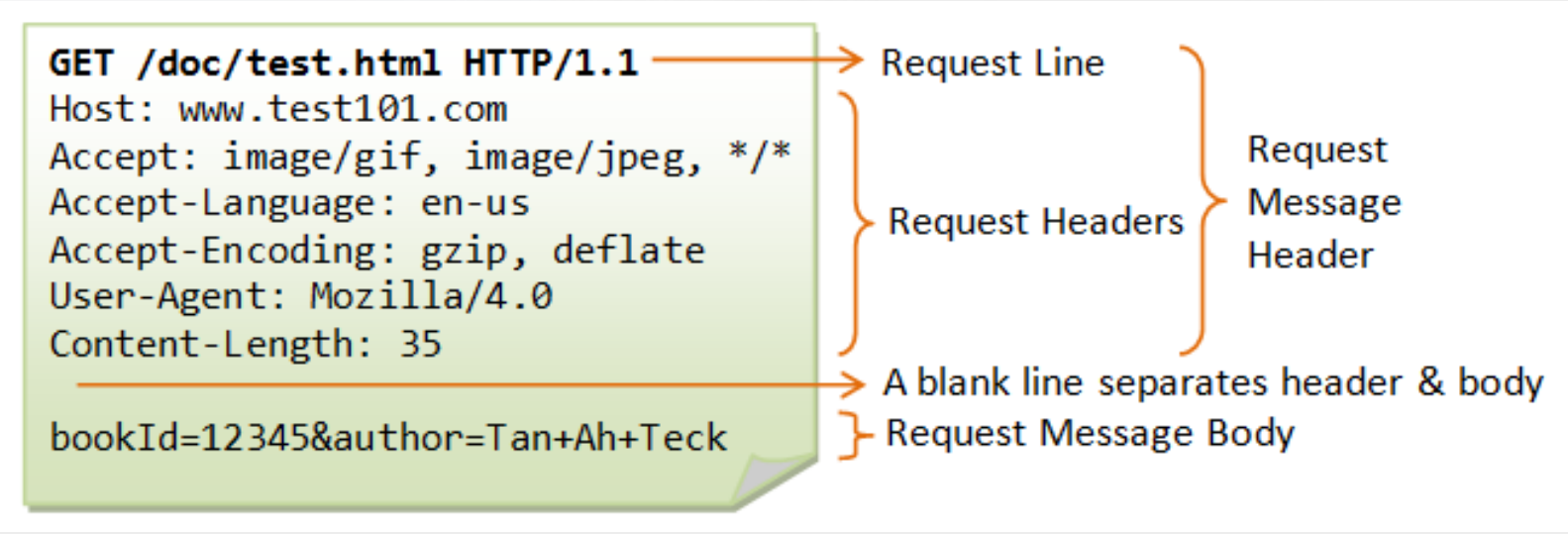
5.1.1 Tìm hiểu về giao thức HTTP

HTTP Requests:
Các phương thức thường dùng

Method	Hoạt động	Chú thích
GET	được sử dụng để lấy lại thông tin từ Server một tài nguyên xác định.	Các yêu cầu sử dụng GET chỉ nên nhận dữ liệu và không nên có ảnh hưởng gì tới dữ liệu
POST	yêu cầu máy chủ chấp nhận thực thể được đính kèm trong request được xác định bởi URI, ví dụ, thông tin khách hàng, file tải lên, ...	
PUT	Nếu URI đề cập đến một tài nguyên đã có, nó sẽ bị sửa đổi; nếu URI không trỏ đến một tài nguyên hiện có, thì máy chủ có thể tạo ra tài nguyên với URI đó.	
DELETE	Xóa bỏ tất cả các đại diện của tài nguyên được chỉ định bởi URI.	
PATCH	Áp dụng cho việc sửa đổi một phần của tài nguyên được xác định.	
...	...	...

5.1.1 Tìm hiểu về giao thức HTTP

HTTP Requests:
Ví dụ



5.1.1 Tìm hiểu về giao thức HTTP

HTTP Responses:
Cấu trúc của một HTTP response:

- **Status-line = Phiên bản HTTP + Mã trạng thái + Trạng thái**
- Có thể có hoặc không có các trường header
- Một dòng trống để đánh dấu sự kết thúc của các trường header
- Tùy chọn một thông điệp

5.1.1 Tìm hiểu về giao thức HTTP

HTTP Responses:

Mã trạng thái: Thông báo về kết quả khi nhận được yêu cầu và xử lí bên server cho client.

Các kiểu mã trạng thái:

1xx: Thông tin (100 -> 101)

VD: 100 (Continue), ….

2xx: Thành công (200 -> 206)

VD: 200 (OK) , 201 (CREATED), …

3xx: Sự điều hướng lại (300 -> 307)

VD: 305 (USE PROXY), …

4xx: Lỗi phía Client (400 -> 417)

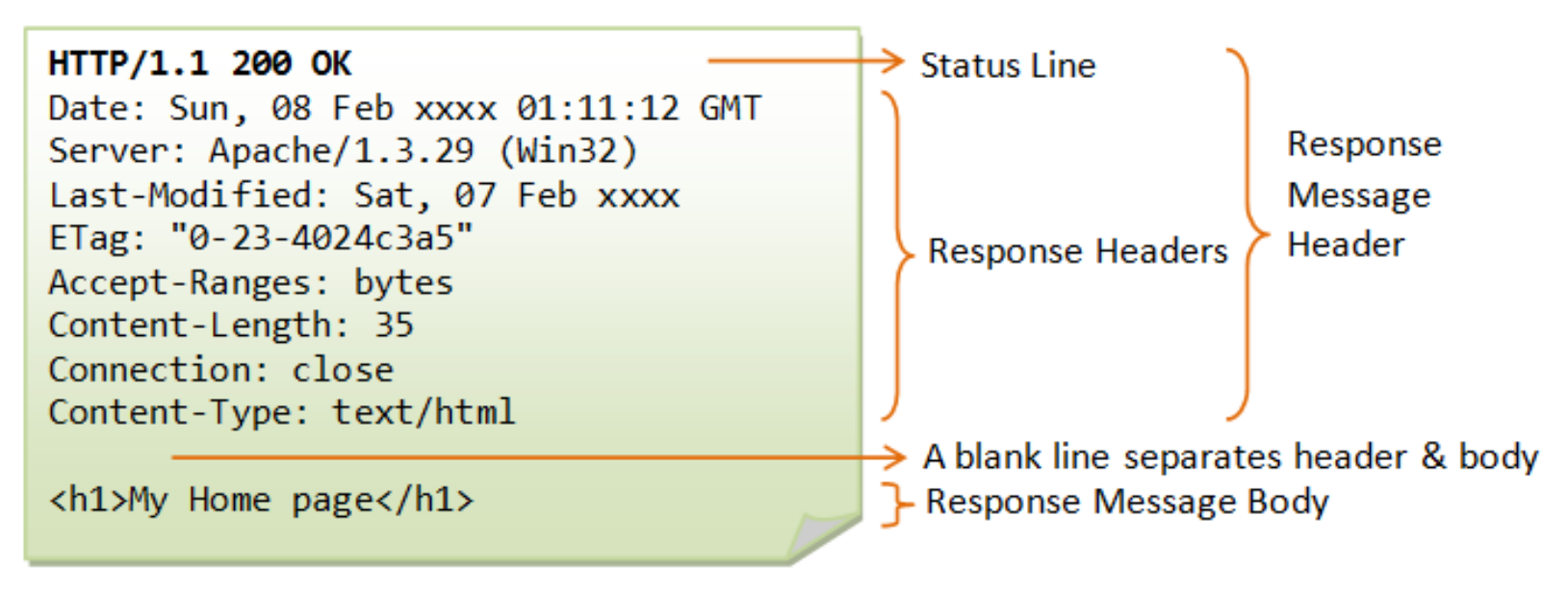
VD: 403 (FORBIDDEN), 404 (NOT FOUND), …

5xx: Lỗi phía Server (500 -> 505)

VD: 500 (INTERNAL SERVER ERROR)

5.1.1 Tìm hiểu về giao thức HTTP

HTTP Responses:
Ví dụ



5.1.2 Một số vấn đề xử lý HTTP request

a) Tách và đọc giá trị các trường header
b) Tách và đọc giá trị chuỗi query string
c) Tách và đọc dữ liệu POST body

a. Tách và đọc giá trị các trường header

- Tách phần header với dòng request và phần body
- Tách từng dòng header (phân cách nhau bởi dấu xuống dòng)
- Với mỗi header, tách phần name và value (phân cách nhau bởi dấu hai chấm)

➢ Sử dụng lệnh strtok() để phân tách chuỗi ký tự
➢ Sử dụng lệnh strstr() để tìm kiếm chuỗi ký tự

a. Tách và đọc giá trị các trường header

```
char* p1 = strstr(req, "\r\n");
char* p2 = strstr(req, "\r\n\r\n");

int headerLength = p2 - (p1 + 2);
char* headers = (char*)malloc(headerLength + 1);
memcpy(headers, p1 + 2, headerLength);
headers[headerLength] = 0;

char* header = strtok(headers, "\r\n");
while (header != NULL)
{
     char* p = strstr(header, ": ");

     int nameLength = p - header;
     char* name = (char*)malloc(nameLength + 1);
     memcpy(name, header, nameLength);
     name[nameLength] = 0;

     int valueLength = strlen(header) - nameLength - 2;
     char* value = (char*)malloc(valueLength + 1);
     memcpy(value, header + nameLength + 2, valueLength);
     value[valueLength] = 0;

     printf("Name: %s --- Value: %s\n", name, value);

     free(name);
     free(value);

     header = strtok(NULL, "\r\n");
}

free(headers);
```

b. Tách và đọc giá trị chuỗi query string

- Tách phần request uri trong dòng request
- Tách chuỗi query string (phía sau dấu ?)
- Tách các cặp tham số trong chuỗi query string (cách nhau bởi dấu &)
- Với mỗi tham số, tách phần name và value (phân cách nhau bởi dấu =)

➢ Sử dụng lệnh strtok() để phân tách chuỗi ký tự
➢ Sử dụng lệnh strstr() để tìm kiếm chuỗi ký tự

c. Tách và đọc dữ liệu POST body

- POST body có các dạng phổ biến:
  - **application/x-www-form-urlencoded:** các tham số được mã hóa dưới dạng key-value (phân cách nhau bởi dấu & và =). Các ký tự không phải chữ cái hoặc chữ số đều được mã hóa nên không phù hợp để truyền dữ liệu nhị phân.
  - **multipart/form-data:** mỗi giá trị được gửi trong một khối dữ liệu ("body part"), phân cách nhau bởi chuỗi ký tự được quy định trong phần header.
  - **text/plain**

c. Tách và đọc dữ liệu POST body

POST body dạng: **application/x-www-form-urlencoded**

```
Received Data: POST / HTTP/1.1
User-Agent: PostmanRuntime/7.25.0
Accept: */*
Postman-Token: 8f18ed28-9752-4fbc-9632-21a2a94d0260
Host: localhost:9000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

username=user1&password=123456&email=user1%40gmail.com
```

c. Tách và đọc dữ liệu POST body

POST body dạng: **application/x-www-form-urlencoded**

```
char* body = strstr(req, "\r\n\r\n") + 4;
if (body != NULL && strlen(body) > 0)
{
    char* param = strtok(body, "&");
    while (param != NULL)
    {
        char* p = strstr(param, "=");
        int nameLength = p - param;
        char* name = (char*)malloc(nameLength + 1);
        memcpy(name, param, nameLength);
        name[nameLength] = 0;
        int valueLength = strlen(param) - nameLength - 1;
        char* value = (char*)malloc(valueLength + 1);
        memcpy(value, param + nameLength + 1, valueLength);
        value[valueLength] = 0;

        printf("Name: %s --- Value: %s\n", name, value);

        free(name);
        free(value);

        param = strtok(NULL, "&");
    }
}
```

c. Tách và đọc dữ liệu POST body

POST body dạng: **multipart/form-data**

```
Received Data: POST / HTTP/1.1
User-Agent: PostmanRuntime/7.25.0
Accept: */*
Postman-Token: 209f061d-437d-4739-9936-c949270f5c6f
Host: localhost:9000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: multipart/form-data; boundary=--------------------------
305312147473165137020446
Content-Length: 400

----------------------------305312147473165137020446
Content-Disposition: form-data; name="username"

user1
----------------------------305312147473165137020446
Content-Disposition: form-data; name="password"

123456
----------------------------305312147473165137020446--
```

c. Tách và đọc dữ liệu POST body

POST body dạng: **multipart/form-data**

```
// Check header Content-Type
if (strstr(req, "Content-Type: multipart/form-data") != NULL)
{
    // Get boundary
    char* p1 = strstr(req, "boundary=") + 9;
    char* p2 = strstr(p1, "\r\n");
    int boundaryLength = p2 - p1 + 2;
    char* boundary = (char*)malloc(boundaryLength + 1);
    strcpy(boundary, "--");
    memcpy(boundary + 2, p1, boundaryLength - 2);
    boundary[boundaryLength] = 0;

    char* body = strstr(req, "\r\n\r\n");
    p1 = strstr(body, boundary) + boundaryLength;
    p2 = strstr(p1, boundary);
    while (p2 != NULL)
    {
        int partLength = p2 - p1;
        char* part = (char*)malloc(partLength + 1);
        memcpy(part, p1, partLength);
        part[partLength] = 0;

        if (strstr(part, "Content-Disposition: form-data;") == NULL)
            continue;
```

c. Tách và đọc dữ liệu POST body

POST body dạng: **multipart/form-data**

```
        char* pp1 = strstr(part, "name=\"") + 6;
        char* pp2 = strstr(pp1, "\"");
        int nameLength = pp2 - pp1;
        char* name = (char*)malloc(nameLength + 1);
        memcpy(name, pp1, nameLength);
        name[nameLength] = 0;

        char* bodyPart = strstr(part, "\r\n\r\n") + 4;

        int valueLength = strlen(part) - (bodyPart - part);
        char* value = (char*)malloc(valueLength + 1);
        memcpy(value, bodyPart, valueLength);
        value[valueLength] = 0;

        printf("Name: %s --- Value: %s\n", name, value);

        free(name);
        free(value);

        p1 = p2 + boundaryLength;
        p2 = strstr(p1, boundary);
    }
    free(boundary);
}
```

5.1.3 Lập trình ứng dụng máy chủ HTTP file

Lập trình ứng dụng máy chủ HTTP file với các chức năng:

- Hiển thị cấu trúc cây thư mục trên máy chủ
- Khi trình duyệt yêu cầu thư mục, hiển thị nội dung của thư mục (thư mục con và files)
- Khi trình duyệt yêu cầu file, trả về nội dung của file, kèm theo kiểu file (Content-Type) và kích thước file (Content-Length)

5.2 Giao thức FTP

5.2.1. Tìm hiểu về giao thức FTP
5.2.2. Lập trình ứng dụng máy chủ FTP
5.2.3. Lập trình ứng dụng máy khách FTP

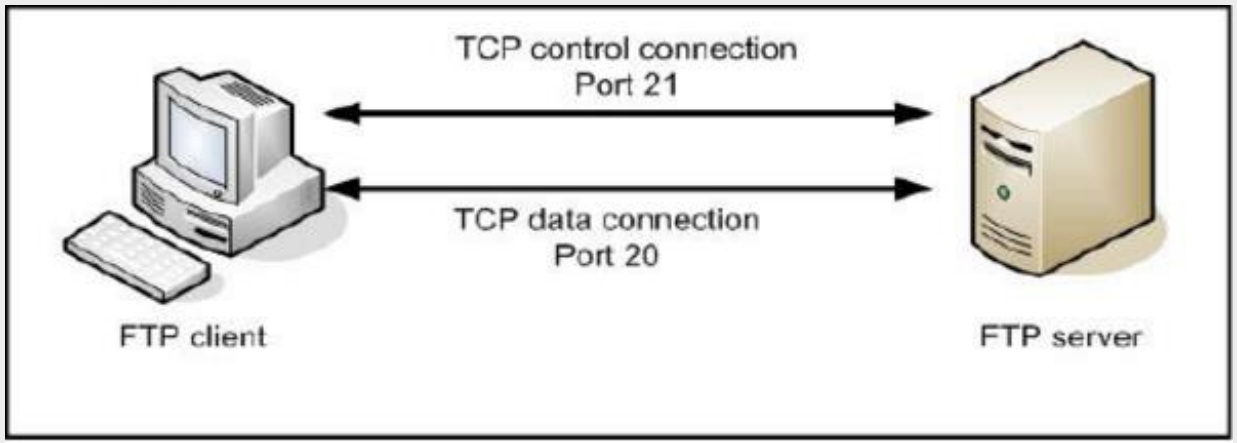
5.2.1 Tìm hiểu giao thức FTP

- Được mô tả trong tài liệu RFC959
- FTP (File Transfer Protocol) là giao thức trao đổi file phổ biến.
- Hoạt động theo mô hình client–server trên nền giao thức TCP.
- Giao diện giữa client và server được cung cấp dưới dạng một tập các lệnh tương tác người dùng.

5.2.1 Tìm hiểu giao thức FTP

Mô hình hoạt động: Quá trình truyền nhận dữ liệu giữa client và server được tạo nên từ 2 tiến trình:

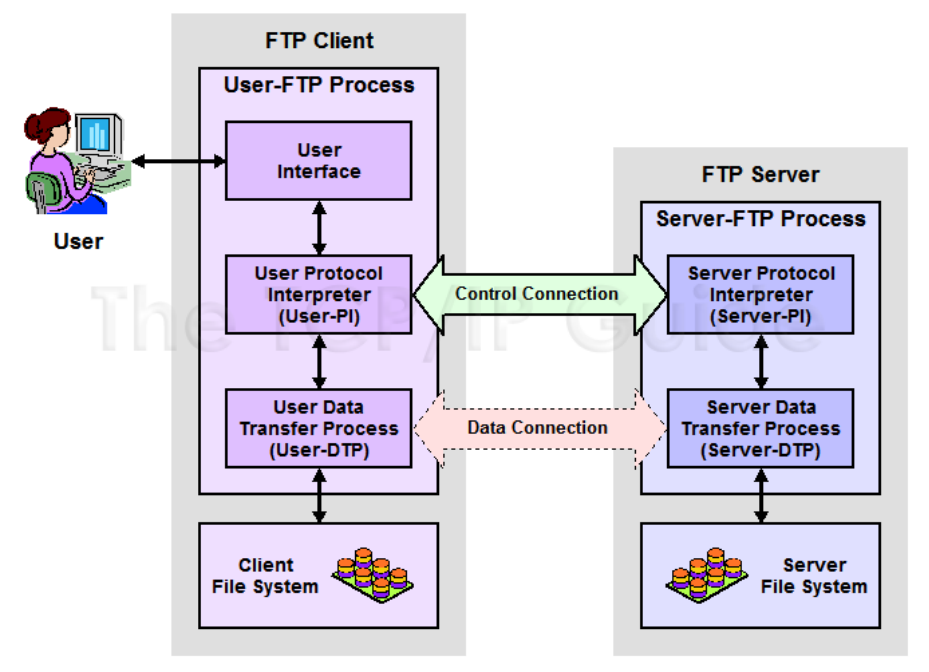
- **Control connection:**
  - Kết nối chính được tạo ra khi phiên làm việc được thiết lập
  - Được duy trì trong suốt phiên làm việc và chỉ cho các thông tin điều khiển đi qua ví dụ như lệnh và trả lời.
  - Không được sử dụng để gửi dữ liệu.
- **Data connection:**
  - Mỗi khi dữ liệu được gửi từ sever tới client hoặc ngược lại, một kết nối dữ liệu được thiết lập. Dữ liệu được truyền qua kết nối này. Khi hoàn tất việc truyền dữ liệu, kết nối được hủy bỏ.



5.2.1 Tìm hiểu giao thức FTP

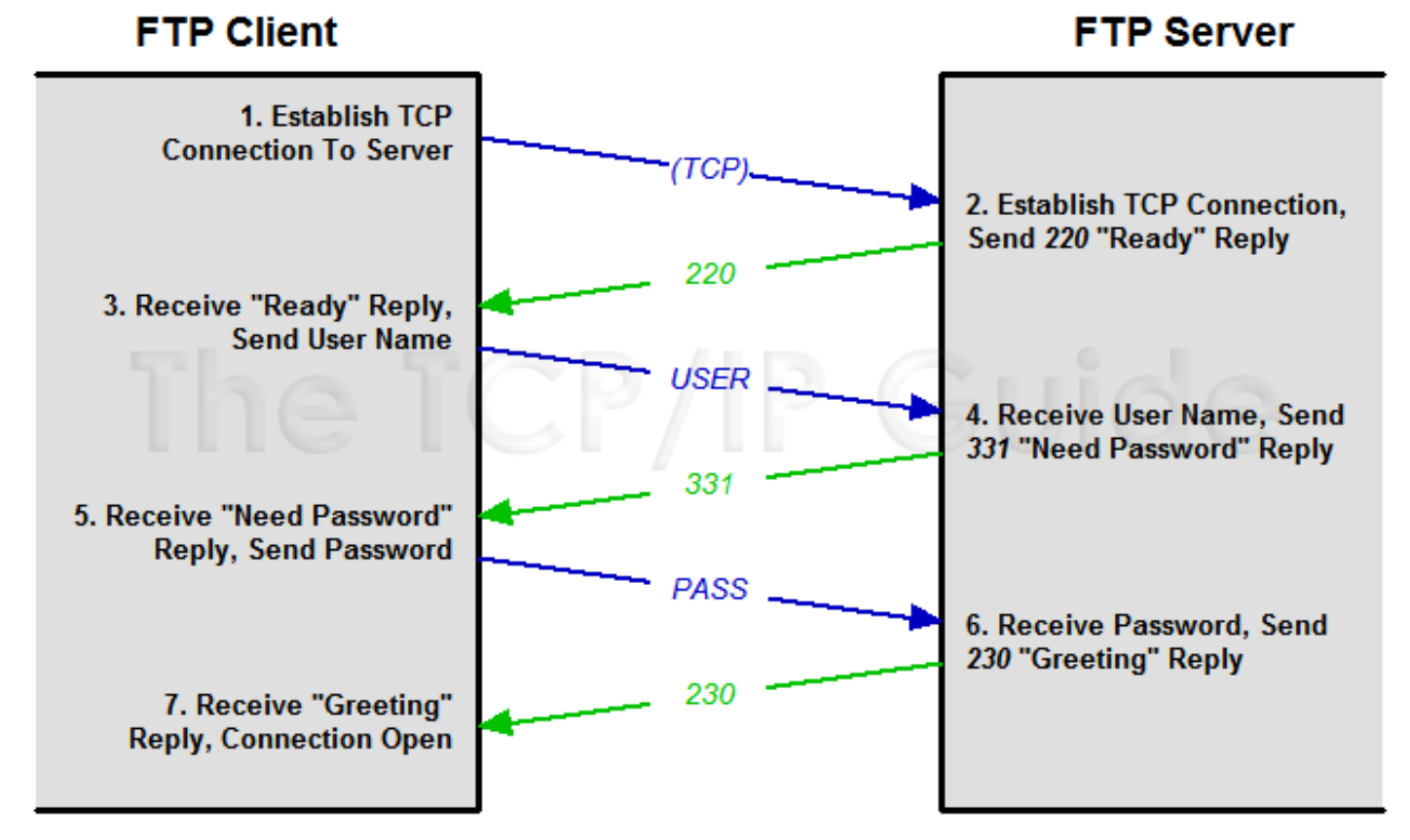
Mô hình hoạt động: mô hình FTP chia phần mềm trên mỗi thiết bị thành 2 thành phần giao thức logic chịu trách nhiệm cho mỗi kênh:

- **Protocol interpreter (PI):** chịu trách nhiệm quản lý kênh điều khiển, phát và nhận lệnh và trả lời.
- **Data transfer process (DTP):** chịu trách nhiệm gửi và nhận dữ liệu giữa client và server.



5.2.1 Tìm hiểu giao thức FTP

Trình tự truy cập và chứng thực FTP: client cung cấp username/password để đăng nhập.



5.2.1 Tìm hiểu giao thức FTP

Quản lý kênh dữ liệu:

- Mỗi khi cần phải truyền dữ liệu giữa các server và client, một kênh dữ liệu cần phải được tạo ra.
- Kênh dữ liệu kết nối bộ phận User-DTP và Server-DTP, sử dụng để truyền file trực tiếp (gửi hoặc nhận một file) hoặc truyền dữ liệu ngầm, như là yêu cầu một danh sách file trong thư mục nào đó trên server.
- Hai phương thức được sử dụng để tạo ra kênh dữ liệu: phía client hay phía server là phía đưa ra yêu cầu khởi tạo kết nối.

5.2.1 Tìm hiểu giao thức FTP

Quản lý kênh dữ liệu:

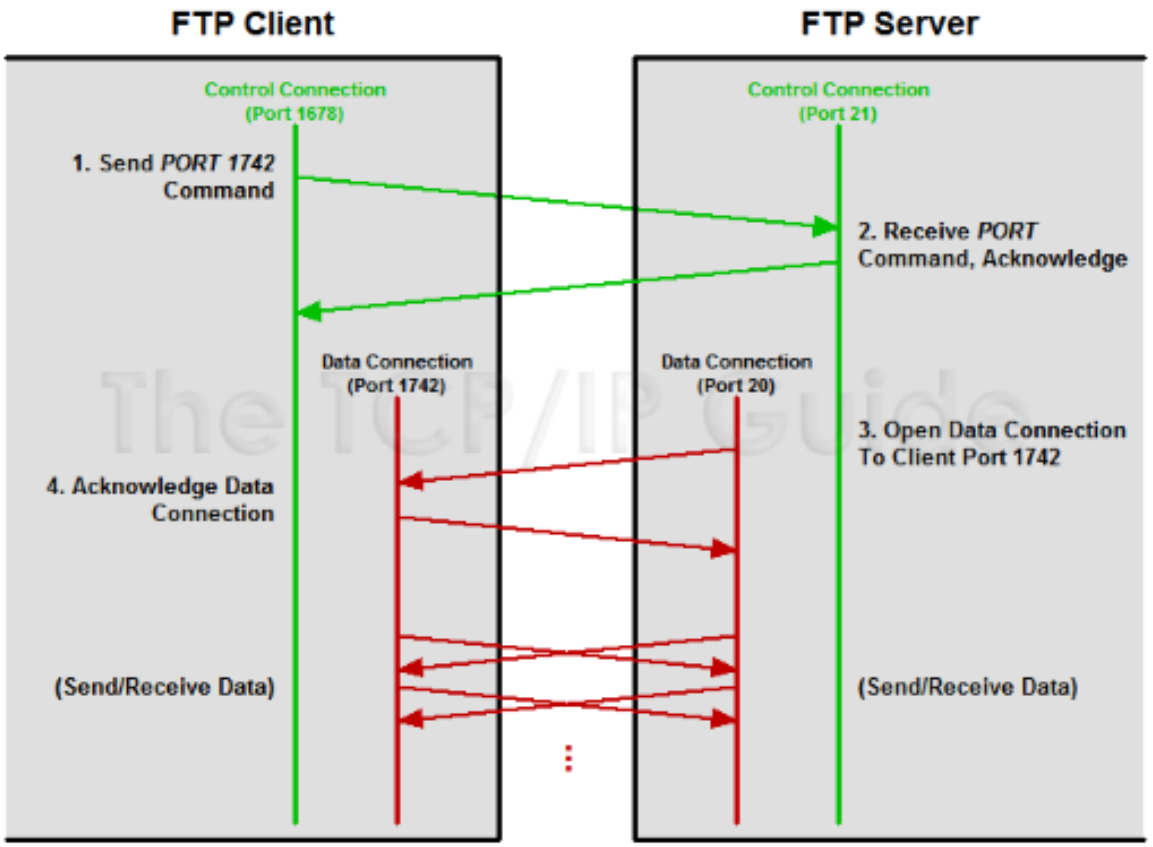


Figure 290: FTP Active Data Connection

5.2.1 Tìm hiểu giao thức FTP

Quản lý kênh dữ liệu: Passive mode

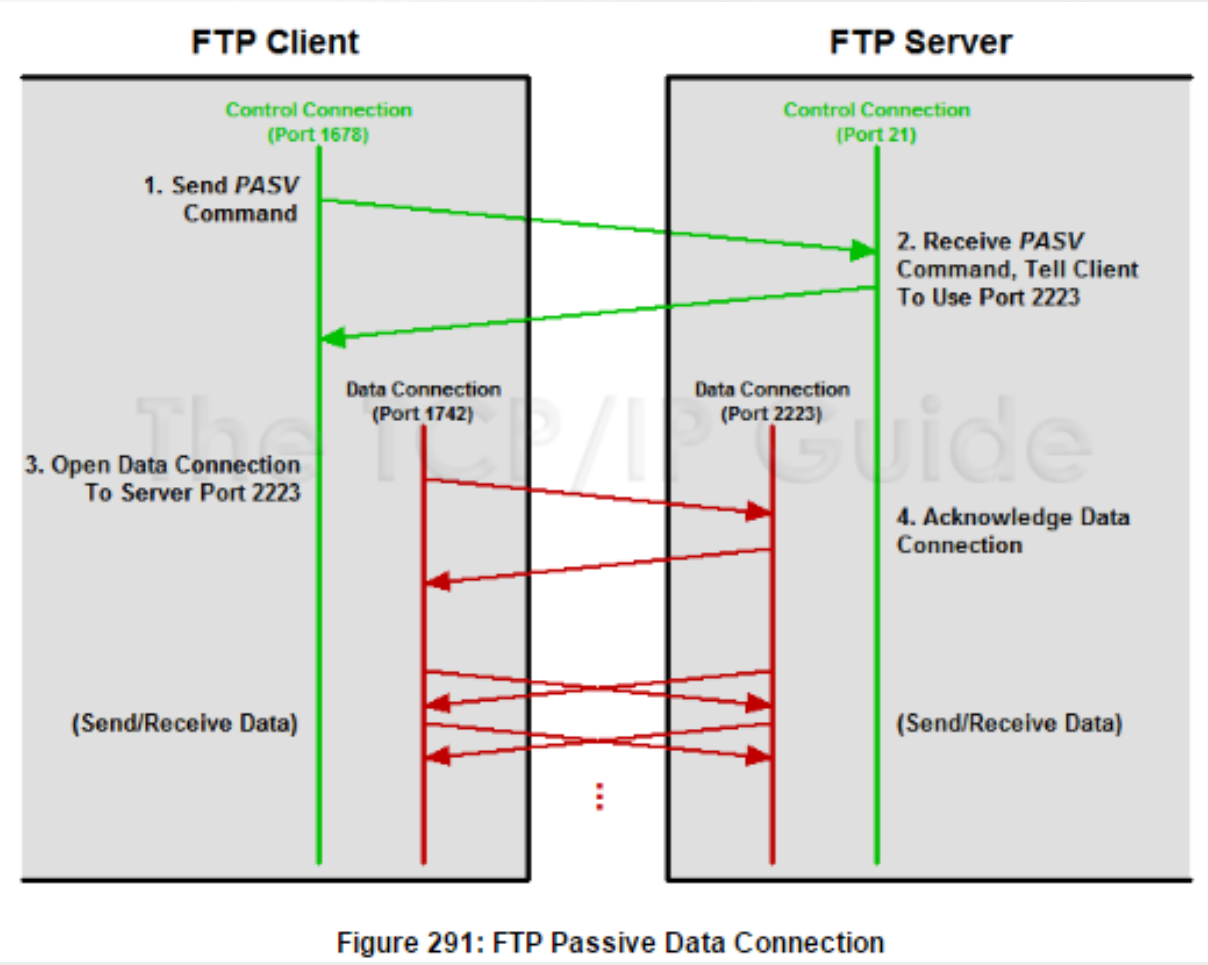


Figure 291: FTP Passive Data Connection

5.2.1 Tìm hiểu giao thức FTP

Một số lệnh FTP thường dùng:

Lệnh + tham số	Ý nghĩa
`USER username`	Gửi định danh người dùng đến server
`PASS password`	Gửi mật khẩu người dùng đến server
`LIST`	Hiển thị danh sách tập tin trong thư mục hiện tại
`RETR filename`	Tải file từ server về client
`STOR filename`	Tải file từ client đến server
`RNFR remote-filename`	Xác định file sẽ được đổi tên
`RNTO remote-filename`	Đổi tên file sang tên mới (sau lệnh RNFR)
`DELE remote-filename`	Xóa tập tin

5.2.1 Tìm hiểu giao thức FTP

Một số lệnh FTP thường dùng:

Lệnh + tham số	Ý nghĩa
`MKD remote-directory`	Tạo thư mục mới
`RMD remote-directory`	Xóa thư mục
`PWD`	In ra tên thư mục hiện tại
`CWD remote-directory`	Di chuyển đến thư mục khác
`TYPE type-character`	Thiết lập kiểu dữ liệu (A: ký tự, I: nhị phân)
`PASV`	Chuyển sang chế độ Passive
`QUIT`	Ngắt kết nối

5.2.2 Lập trình ứng dụng máy chủ FTP

Lập trình ứng dụng máy chủ FTP với các chức năng cơ bản:

- Quản lý người dùng
- Đăng nhập
- Hiển thị nội dung thư mục
- Tạo, đổi tên, xóa thư mục
- Đổi tên, xóa file
- Client tải file lên server (upload)
- Client tải file từ server (download)

5.2.3 Lập trình ứng dụng máy khách FTP

Lập trình ứng dụng máy khách FTP với các chức năng cơ bản:

- Cung cấp giao diện cơ bản (người dùng không trực tiếp nhập các lệnh FTP)
- Đăng nhập
- Hiển thị nội dung thư mục
- Tạo, đổi tên, xóa thư mục
- Đổi tên, xóa file
- Client tải file lên server (upload)
- Client tải file từ server (download)

5.3 Giao thức POP3

5.3.1. Tìm hiểu về giao thức POP3
5.3.2. Lập trình ứng dụng máy khách POP3

5.3.1 Tìm hiểu giao thức POP3

- Post Office Protocol phiên bản 3.
- Được mô tả trong tài liệu RFC1939
- Giao thức ở tầng ứng dụng được sử dụng để lấy thư điện tử từ server mail, thông qua kết nối TCP/IP.
- Giao diện giữa client và server được cung cấp dưới dạng một tập các lệnh tương tác người dùng.

5.3.1 Tìm hiểu giao thức POP3

- Thường hoạt động ở cổng 110.
- Các trạng thái hoạt động trên server với mỗi kết nối:
  - AUTHORIZATION
  - TRANSACTION
  - UPDATE

5.3.1 Tìm hiểu giao thức POP3

Một số lệnh POP3 thường dùng:

Trạng thái	Lệnh + tham số	Ý nghĩa
AUTHORIZATION	`USER username`	Gửi định danh người dùng đến server
	`PASS password`	Gửi mật khẩu người dùng đến server
TRANSACTION	`STAT`	Hiển thị thông tin hộp thư (số thư + kích thước)
	`LIST [msg]`	Liệt kê các thư và kích thước
	`RETR msg`	Hiển thị nội dung thư
	`DELE msg`	Đánh dấu thư sẽ bị xóa
	`NOOP`	Giữ trạng thái kết nối
	`RSET`	Khôi phục trạng thái đánh dấu thư bị xóa
UPDATE	`QUIT`	Xóa các thư đã đánh dấu, ngắt kết nối

5.3.2 Lập trình ứng dụng máy khách POP3

Lập trình ứng dụng máy khách POP với các chức năng cơ bản:

- Cung cấp giao diện cơ bản (người dùng không trực tiếp nhập các lệnh POP3)
- Đăng nhập
- Hiển thị danh sách email
- Hiển thị nội dung email

Phụ lục 1: Raw Sockets

- Raw sockets cho phép truy nhập vào các giao thức ở tầng giao vận (Transport protocol).
- Raw sockets có thể được sử dụng để tạo ra những tiện ích như ứng dụng Ping, sniffer
- Cần có hiểu biết cơ bản về những giao thức như ICMP, TCP, …

Raw Sockets

- Khởi tạo raw sockets:
  - Sử dụng hàm socket() hoặc WSASocket()
  - Ví dụ:

❖ Tạo raw socket để bắt các gói tin IP:

```
SOCKET s1 = socket(AF_INET, SOCK_RAW, IPPROTO_IP);
if (s1 == INVALID_SOCKET)
{
    printf("Failed to create socket\n");
    return 1;
}
```

❖ Tạo raw socket truyền gói tin ICMP:

```
SOCKET s2 = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

Raw Sockets

- **Ví dụ 1** - Ứng dụng sniffer
  - Sử dụng raw socket để bắt các gói tin IP bao gồm cả header và data
  - Raw socket được gắn với giao diện mạng nào thì nó sẽ bắt các gói tin truyền qua giao diện mạng đó
  - Cần sử dụng hàm **WSAIoctl()** với tham số **SIO_RCVALL** để socket có thể nhận được các gói tin IP
  - Sử dụng hàm **recvfrom()** để nhận các gói tin.
  - Mỗi lần gọi hàm **recvfrom()** với độ lớn buffer bằng chiều dài tối đa của gói tin IP (65536 bytes)

➢ *Cần chạy Visual Studio hoặc chương trình với quyền Administrator để có thể tạo raw sockets*

Raw Sockets

- **Ví dụ 1** - Ứng dụng sniffer

```
#include "stdafx.h"

#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include "winsock2.h"
#include "mstcpip.h"

int main()
{
    WSADATA wsa;
    WSAStartup(MAKEWORD(2, 2), &wsa);

    SOCKET sniffer = socket(AF_INET, SOCK_RAW, IPPROTO_IP);
    if (sniffer == INVALID_SOCKET)
    {
        printf("Failed to create socket\n");
        system("pause");
        return 1;
    }
```

Raw Sockets

- **Ví dụ 1** - Ứng dụng sniffer

```
SOCKADDR_IN addr;
addr.sin_family = AF_INET;
addr.sin_port = 0;
addr.sin_addr.s_addr = inet_addr("192.168.15.11");

bind(sniffer, (SOCKADDR *)&addr, sizeof(addr));

int opt = RCVALL_ON;
int bytesReturned = 0;
if (WSAIoctl(sniffer, SIO_RCVALL, &opt, sizeof(opt), 0, 0,
    (LPDWORD)&bytesReturned, 0, 0) == SOCKET_ERROR)
{
    printf("WSAIoctl() failed.\n");
    return 1;
}

char *buf = (char *)malloc(65536);
int res;
unsigned char ip_protocol;
```

Raw Sockets

- **Ví dụ 1** - Ứng dụng sniffer

```
    while (1)
    {
        res = recvfrom(sniffer, buf, 65536, 0, 0, 0);
        if (res <= 0) break;

        // Kiem tra loai protocol
        memcpy(&ip_protocol, buf + 9, 1);
        if (ip_protocol == 1)
            printf("ICMP\n");
        else if (ip_protocol == 6)
            printf("TCP\n");
        else if (ip_protocol == 17)
            printf("UDP\n");
    }

    free(buf);
    closesocket(sniffer);
    WSACleanup();
    return 0;
}
```

Raw Sockets

- **Ví dụ 2** - Ứng dụng ping sử dụng giao thức ICMP
  - Sử dụng raw socket để bắt các gói tin IP bao gồm cả header và data
  - Raw socket được gắn với giao diện mạng nào thì nó sẽ bắt các gói tin truyền qua giao diện mạng đó
  - Cần sử dụng hàm **WSAIoctl()** với tham số **SIO_RCVALL** để socket có thể nhận được các gói tin IP
  - Sử dụng hàm **recvfrom()** để nhận các gói tin.
  - Mỗi lần gọi hàm **recvfrom()** với độ lớn buffer bằng chiều dài tối đa của gói tin IP (65536 bytes)

➢ *Cần chạy Visual Studio hoặc chương trình với quyền Administrator để có thể tạo raw sockets*

Raw Sockets

- **Bài tập 1:** Chỉnh sửa ví dụ 1 để có thể
  - Hiển thị kiểu giao thức cùng với địa chỉ IP nguồn và đích
  - Với gói tin TCP, kiểm tra xem có phải là lệnh GET hoặc lệnh POST hay không?

Phụ lục 2. Thư viện OpenSSL

a. Giới thiệu
b. Sử dụng công cụ Command Line
c. Cách cài đặt vào dự án
d. Một số ví dụ

a. Giới thiệu

- OpenSSL là bộ công cụ mạnh và đầy đủ tính năng phục vụ cho giao thức TLS (Transport Layer Security) và SSL (Secure Sockets Layer).
- OpenSSL cung cấp thư viện bảo mật đa mục đích.
- Mã nguồn được tải miễn phí từ địa chỉ:
  https://www.openssl.org/source/
- Công cụ đã biên dịch và thư viện cho Windows được tải về từ địa chỉ:
  https://slproweb.com/products/Win32OpenSSL.html

b. Sử dụng công cụ Command Line

- Sử dụng Command Prompt truy nhập đến thư mục chứa chương trình **openssl.exe**, thường là
  C:\Program Files\OpenSSL-Win64\bin

- Ví dụ: Thực hiện kết nối qua giao thức HTTPS
  openssl s_client vnexpress.net:443

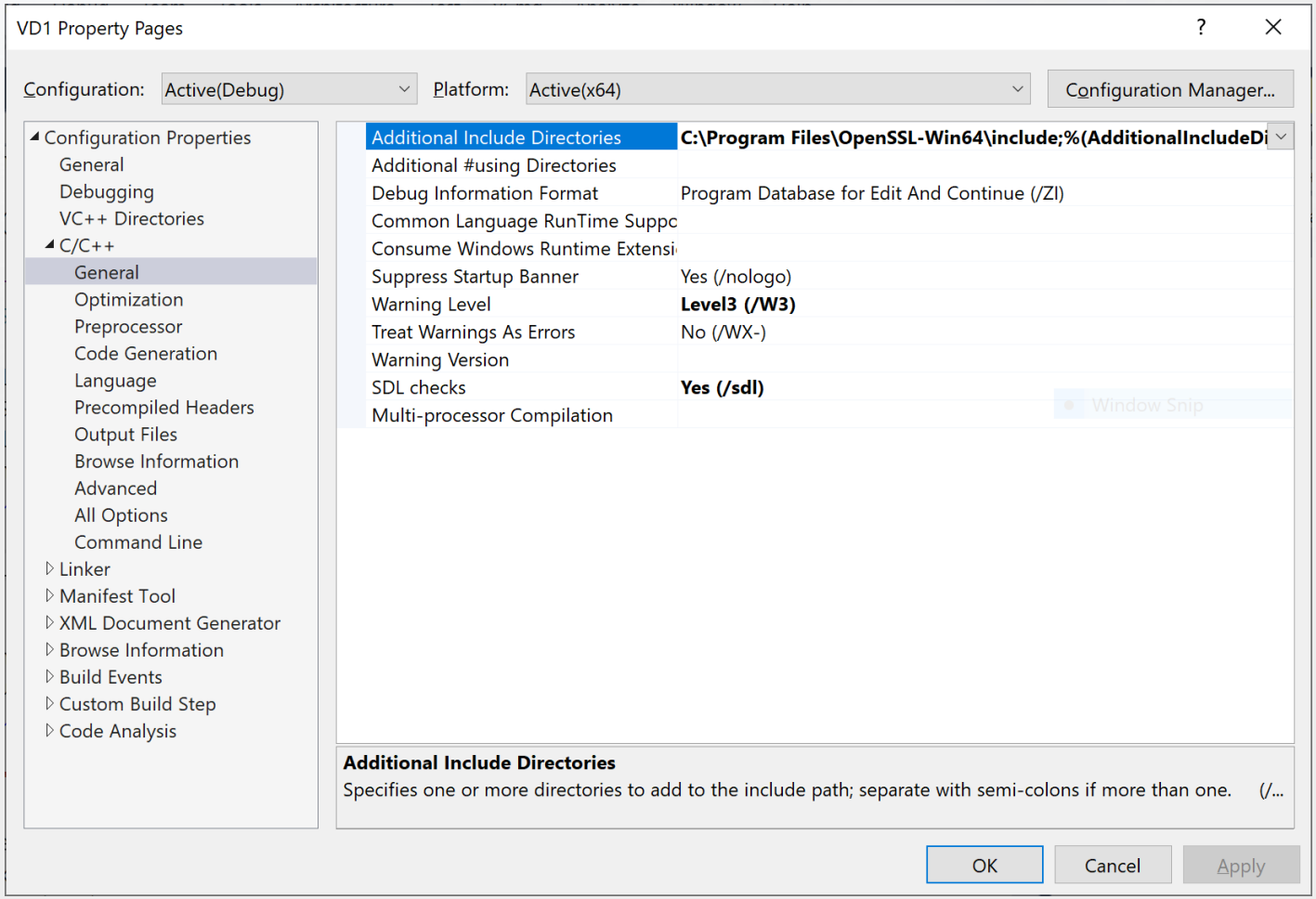
=> sau khi kết nối thành công, sử dụng phương thức GET để nhận dữ liệu

c. Cách cài đặt vào dự án

- Chọn và cài đặt thư viện OpenSSL đúng với phiên bản của dự án (32 hoặc 64 bits)
- Thêm thư mục chứa tệp tiêu đề
  - Đường dẫn tới thư mục chứa tệp tiêu đề thường là:
    C:\Program Files\OpenSSL-Win64\include
  - Chọn **Project Properties** => **C/C++** => **General**
  - Thêm đường dẫn vào **Additional Include Directories**

c. Cách cài đặt vào dự án

- Thêm thư mục chứa tệp tiêu đề



c. Cách cài đặt vào dự án

- Thêm thư mục chứa tệp thư viện
  - Đường dẫn tới thư mục chứa tệp thư viện thường là:
    C:\Program Files\OpenSSL-Win64\lib\VC
  - Chọn **Project Properties => Linker => General**
  - Thêm đường dẫn vào **Additional Library Directories**
- Khai báo tệp thư viện
  - Chọn **Project Properties => Linker => Input**
  - Thêm thư viện **libssl64MD.lib** và **libssl64MT.lib** vào **Additional Dependencies**

c. Cách cài đặt vào dự án

- Khởi tạo thư viện

```
SSL_library_init();           // Khởi tạo thư viện OpenSSL
const SSL_METHOD *meth = TLS_client_method(); // Khai báo phương thức mã hóa TLS
SSL_CTX *ctx = SSL_CTX_new(meth);   // Tạo context mới
SSL *ssl = SSL_new(ctx);      // Tạo đối tượng ssl
if (!ssl) {
    printf("Error creating SSL.\n");
    return -1;
}
SSL_set_fd(ssl, client);      // Gắn đối tượng ssl với socket
int err = SSL_connect(ssl);   // Tạo kết nối ssl
if (err <= 0) {
    printf("Error creating SSL connection.  err=%x\n", err);
    fflush(stdout);
    return -1;
}
```

c. Cách cài đặt vào dự án

- Truyền nhận dữ liệu thông qua đối tượng ssl:

Sau bước khởi tạo, đối tượng **ssl** được sử dụng để truyền nhận dữ liệu mã hóa

Nhận dữ liệu:

```
int SSL_read(SSL *ssl, void *buf, int num)
```

ssl: con trỏ đối tượng ssl đã khởi tạo
buf: buffer nhận dữ liệu
num: số byte muốn nhận
=> Hàm trả về số byte nhận được trong trường hợp thành công

Truyền dữ liệu:

```
int SSL_write(SSL *ssl, const void *buf, int num)
```

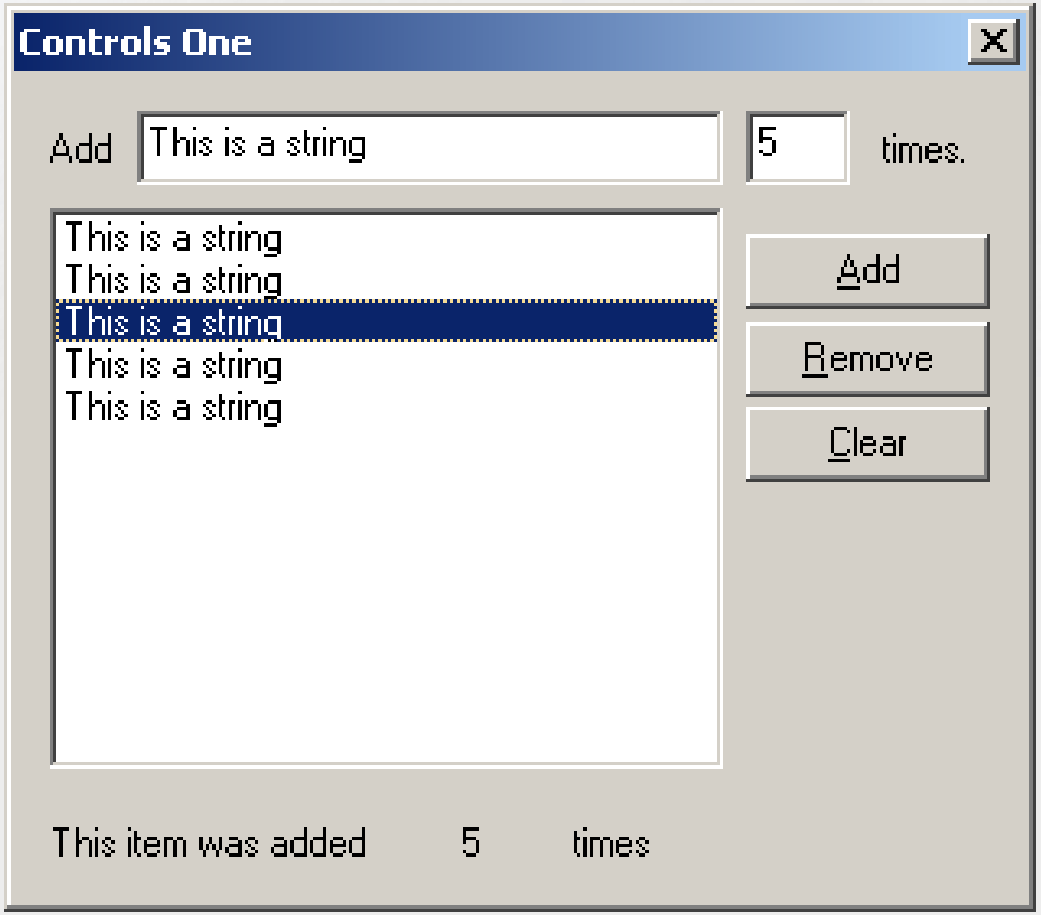
ssl: con trỏ đối tượng ssl đã khởi tạo
buf: buffer chứa dữ liệu muốn truyền
num: số byte cần truyền
=> Hàm trả về số byte truyền được trong trường hợp thành công

d. Một số ví dụ

- Kết nối và nhận dữ liệu từ website HTTPS
- Kết nối đến Gmail server thông qua phương thức POP3

Phụ lục 3. Một số đối tượng giao diện

- ListBox
- EditText
- Button



Đối tượng BUTTON

- Tạo mới

```
CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("BUTTON"), TEXT("OK"),
WS_CHILD | WS_VISIBLE | WS_TABSTOP | BS_DEFPUSHBUTTON, 420,
360, 150, 40, hWnd, (HMENU)IDC_BUTTON_OK,
GetModuleHandle(NULL), NULL);
```

- Sự kiện khi nhấn nút được xử lý thông qua hàm xử lý sự kiện của cửa sổ

Đối tượng LISTBOX

- Tạo mới

```
CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("LISTBOX"), TEXT(""),
WS_CHILD | WS_VISIBLE | WS_TABSTOP | ES_AUTOVSCROLL, 10, 10,
160, 350, hWnd, (HMENU)IDC_LIST_CLIENT, GetModuleHandle(NULL),
NULL);
```

- Thêm dòng mới vào ListBox

```
SendDlgItemMessageA(hWnd, IDC_LIST_CLIENT, LB_ADDSTRING, 0,
(LPARAM) "Hello");
```

- Cuộn ListBox xuống dưới

```
SendDlgItemMessageA(hWnd, IDC_LIST_CLIENT, WM_VSCROLL,
SB_BOTTOM, 0);
```

- Lấy chỉ số của dòng đang được chọn

```
int i = SendDlgItemMessageA(hWnd, IDC_LIST_CLIENT, LB_GETCURSEL,
0, 0);
```

Đối tượng EDIT

- Tạo mới

```
CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("EDIT"), TEXT(""),
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 10, 360, 400, 40, hWnd,
(HMENU)IDC_EDIT_TEXT, GetModuleHandle(NULL), NULL);
```

- Lấy dữ liệu

```
GetDlgItemTextA(hWnd, IDC_EDIT_TEXT, buf, sizeof(buf));
```

- Thiết lập dữ liệu

```
SetDlgItemTextA(hWnd, IDC_EDIT_TEXT, "");
```

Phụ lục 4. Sử dụng hàm Windows API để duyệt danh sách thư mục/file

- Sử dụng hàm FindFirstFileA() và FindNextFileA() để lấy danh sách thư mục và tập tin của một thư mục bất kỳ.
- Cấu trúc WIN32_FIND_DATAA được sử dụng để lưu dữ liệu trả về bởi 2 hàm trên.

```
typedef struct _WIN32_FIND_DATA {
  DWORD    dwFileAttributes;              // Thuộc tính file
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD    nFileSizeHigh;                 // Kích thước file
  DWORD    nFileSizeLow;                  // Kích thước file
  DWORD    dwReserved0;
  DWORD    dwReserved1;
  TCHAR    cFileName[MAX_PATH];           // Tên file
  TCHAR    cAlternateFileName[14];
} WIN32_FIND_DATA, *PWIN32_FIND_DATA, *LPWIN32_FIND_DATA;
```

Sử dụng hàm Windows API để duyệt danh sách thư mục / file

- Các trường cần quan tâm gồm **dwFileAttributes** chứa thuộc tính của thư mục/tập tin, được sử dụng để phân biệt thư mục hoặc tập tin, **cFileName** chứa tên của thư mục/tập tin, **nFileSizeHigh** và **nFileSizeLow** chứa kích thước của tập tin.
- Kích thước của file được xác định theo công thức:

```
(nFileSizeHigh*(MAXDWORD+1))+nFileSizeLow
```

Sử dụng hàm Windows API để duyệt danh sách thư mục / file

```
WIN32_FIND_DATAA DATA;
HANDLE h = FindFirstFileA("C:\\*.*", &DATA);
do {
   if (DATA.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
   {
      // Đây là một thư mục
      // In ra tên thư mục
   }
   else
   {
      // Đây là một file
      // In ra tên file và kích thước file
   }
} while (FindNextFileA(h, &DATA));
```