

LẬP TRÌNH MẠNG

Network Programming

Giảng viên: Lê Bá Vui

Email: vuilb@soict.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.
- 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++**, **C#**
 - Các kỹ thuật lập trình: mảng, chuỗi ký tự, 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 + BTVN
- **Điểm cuối kỳ** = Điểm thi cuối kỳ + Bài tập lớn + Thực hành

Thời lượng môn học

- Thời lượng: 45 tiết
 - Lý thuyết + Bài tập: 39 tiết
 - Trình bày Bài tập lớn: 6 tiết

Tài liệu tham khảo

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

Link tải bài giảng
<https://goo.gl/Kktm59>

Nội dung môn học

Chương 1. Giới thiệu các mô hình lập trình mạng

Chương 2. Bộ giao thức TCP/IP

Chương 3. Windows Socket

Chương 4. MFC Socket

Chương 5. .NET Socket

Chương 1. Giới thiệu các mô hình Lập trình mạng

Chương 1. Giới thiệu các mô hình lập trình mạng

1.1. Tổng quan về lập trình mạng

1.2. Các đề tài Bài tập lớn

1.1. Tổng quan về lập trình mạng

- **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. Tổng quan về lập trình mạng

- 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. Tổng quan về lập trình mạng

- 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++** và **C#**.

1.1. Tổng quan về lập trình mạng

- Các 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. Tổng quan về lập trình mạng

- Ví dụ về các ứng dụng trên mạng
 - Phần mềm nghe nhạc trên thiết bị di động
 - Máy chủ quản lý dữ liệu người dùng, 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. Tổng quan về lập trình mạng

- Các 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
 - **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. Tổng quan về lập trình mạng

- Các thư viện được sử dụng:
 - **MFC Socket**
 - Nằm trong bộ thư viện MFC của Microsoft
 - Đóng gói các hàm của WinSock dưới dạng các lớp hướng đối tượng
 - Dễ sử dụng và hiệu năng cao
 - Thư viện của các ngôn ngữ khác: Java, PHP, Python...

⇒ Thư viện sử dụng trong giáo trình: **WinSock, MFC Socket, System.Net và System.Net.Sockets**

1.1. Tổng quan về lập trình mạng

- Các công cụ lập trình
 - **Visual Studio (2015, 2017, ...)**
 - Rất mạnh
 - Hỗ trợ cả WinSock, MFC Socket và .NET Socket
 - Bản Community được tải miễn phí
 - **Dev C++**
 - Miễn phí
 - Chỉ hỗ trợ WinSock

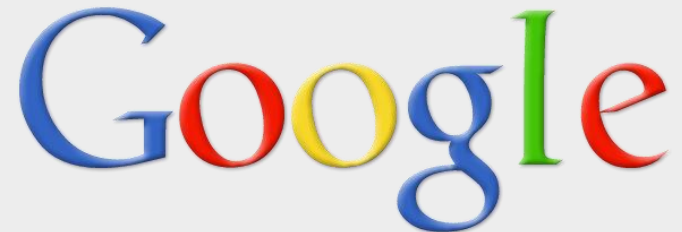


1.1. Tổng quan về lập trình mạng

- 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. Tổng quan về lập trình mạng

- 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. Bài tập lớn môn học

- **Bài tập lớn môn học (yêu cầu chung)**
 - Chia thành các nhóm, mỗi nhóm không quá 4 người, phân chia công việc rõ ràng. Tổ chức theo các module độc lập.
 - Mỗi nhóm thực hiện một đề tài. Có thể chọn các đề tài có sẵn hoặc tự đề xuất (cần có sự đồng ý của giáo viên)
 - Thang điểm tối đa 10. Được sử dụng để xét điểm cuối kỳ.
 - Trình bày trên lớp:
 - Lần 1: Trình bày tổng quan về đề tài đã chọn, các vấn đề lý thuyết đã tìm hiểu. Phân chia công việc các thành viên trong nhóm.
 - Lần 2: Trình bày cụ thể các vấn đề kỹ thuật, kết quả thực hiện chương trình, demo minh họa.

1.2. Bài tập lớn môn học

- **Đề tài: 01 – FTP Client**

Viết chương trình FTP Client thực hiện được ít nhất các chức năng sau:

- Đăng nhập vào một máy chủ FTP
- Hiển thị danh sách các thư mục, tập tin trên máy chủ
- Download một tập tin được chọn, download nhiều tập tin đồng thời, download thư mục
- Upload tập tin được chọn, upload nhiều tập tin đồng thời, upload thư mục
- Đổi tên, xóa một tập tin, thư mục được chọn
- Tạo một thư mục mới
- **Ngôn ngữ C/C++**
- **Tự lập trình mức SOCKET**
- Độ khó: 2

1.2. Bài tập lớn môn học

- **Đề tài: 02 – FTP Server**

Viết chương trình FTP Server thực hiện được ít nhất các chức năng sau:

- Hỗ trợ các lệnh cơ bản: đăng nhập, upload, download tập tin, tạo mới, đổi tên, xóa tập tin, thư mục, ...
- Quản lý người dùng: thêm, xóa, thay đổi thông tin (username và password)
- Hỗ trợ nhiều kết nối đồng thời
- **Ngôn ngữ C/C++**
- **Tự lập trình mức SOCKET**
- Độ khó: 3

1.2. Bài tập lớn môn học

- **Đề tài: 03 – HTTP Downloader**

Viết chương trình tải tệp tin theo giao thức HTTP thực hiện được ít nhất các chức năng sau:

- Tải tệp tin ở một địa chỉ xác định
- Tải file bằng nhiều luồng song song
- Hỗ trợ tính năng Pause/Resume
- Tải nhiều tệp tin từ 1 danh sách (tệp tin văn bản)
- **Ngôn ngữ C/C++**
- **Tự lập trình mức SOCKET**
- Độ khó: 3

1.2. Bài tập lớn môn học

- **Đề tài: 04 – Web Server**

Viết chương trình tạo một Web Server thực hiện được ít nhất các công việc sau:

- Quản lý người dùng (lưu trữ trong file văn bản)
- Sử dụng template là các trang HTML
- Trang đăng ký, đăng nhập
- Hiển thị danh sách các tài khoản đã đăng ký
- Thay đổi thông tin của tài khoản được chọn
- Đáp ứng được nhiều kết nối đồng thời
- Thống kê và ghi nhật ký (địa chỉ IP, thời gian, lệnh, ...)
- **Ngôn ngữ C/C++**
- **Tự lập trình mức SOCKET**
- Độ khó: 3

1.2. Bài tập lớn môn học

- **Đề tài: 05 – Dò và phát hiện mp3 links**
 - Sử dụng thư viện winpcap hoặc một thư viện tương đương
 - Lập trình để bắt các gói tin.
 - Phân tích nội dung gói tin để phát hiện link mp3 (tương tự như IDM)
 - Tải về client link mp3 đã tìm được
 - **Ngôn ngữ C/C++**
 - **Tự lập trình mức SOCKET**
 - Độ khó: 3

1.2. Bài tập lớn môn học

- **Đề tài: 06 – Simple Email Client**
 - Viết chương trình kết nối và đăng nhập vào một POP3 server (Gmail)
 - Hiển thị các email và trạng thái tương ứng (đã đọc, chưa đọc...)
 - Lấy dữ liệu email và hiển thị
 - Tải và lưu trữ các file đính kèm nếu có
 - **Ngôn ngữ: C/C++**
 - **Lập trình mức SOCKET**
 - Độ khó: 3

1.2. Bài tập lớn môn học

- **Đề tài: 07 – Chat Voice**
 - Ưu tiên thiết lập kết nối trực tiếp giữa các client
 - Trong trường hợp không thể thiết lập kết nối trực tiếp thì có thể trung gian qua server
 - Giao thức tự thiết kế, codec tự chọn (mp3, wma, aac, ...). Không truyền âm thanh chưa nén (wav, pcm).
 - Mã hóa kênh truyền theo giải thuật tùy chọn.
 - Ngôn ngữ C/C++/MFC.
 - Cho phép sử dụng OpenSource lame để nén và giải nén MP3.
 - Độ khó: 5

1.2. Bài tập lớn môn học

- **Đề tài: 08 – Gomoku Game Server & Client**
 - Xây dựng hệ thống gồm server và client
 - Quản lý người dùng (lưu trữ trong file văn bản)
 - Tự thiết kế giao thức giữa client và server
 - Hỗ trợ đăng ký, đăng nhập
 - Tạo ván chơi giữa 2 người chơi đã đăng nhập
 - Theo dõi các nước đi, kiểm tra trạng thái của ván chơi
 - Lưu nhật ký các ván chơi
 - **Ngôn ngữ C/C++**
 - **Tự lập trình mức SOCKET**
 - Độ khó: 4

1.2. Bài tập lớn môn học

- **Đề tài khác:**

- **Remote Desktop:** điều khiển máy tính từ xa, điều khiển chuột, bàn phím, hiển thị hình ảnh
- **Image Streaming:** Client thu thập hình ảnh qua webcam và truyền về Server
- **Radio Broadcast:** quản lý các kênh phát âm thanh theo thời gian thực:
 - Nguồn phát: liên tục phát dữ liệu âm thanh đến địa chỉ multicast
 - Nguồn thu: nhận dữ liệu âm thanh từ địa chỉ multicast và phát âm thanh qua loa
 - Server: quản lý kết nối giữa nguồn phát và nguồn thu
- **Web Proxy Server:** quản lý các kết nối web

1.2. Bài tập lớn môn học

- **Đề tài tự chọn:**
 - Sinh viên có thể tùy chọn đề tài
 - Cần được sự đồng ý giáo viên trước khi thực hiện
 - **Ngôn ngữ: C/C++**
 - **Lập trình mức SOCKET**

Chương 2. Bộ giao thức Internet TCP/IP

Chương 2. Bộ giao thức Internet (TCP/IP)

2.1. Giới thiệu

2.2. Giao thức IPv4

2.3. Giao thức IPv6

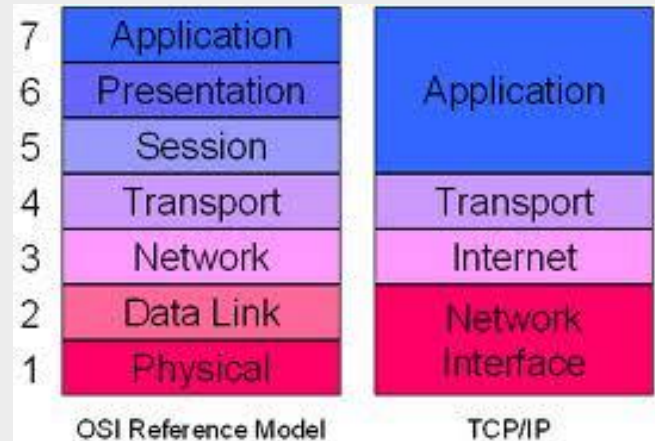
2.4. Giao thức TCP

2.5. Giao thức UDP

2.6. Hệ thống phân giải tên miền

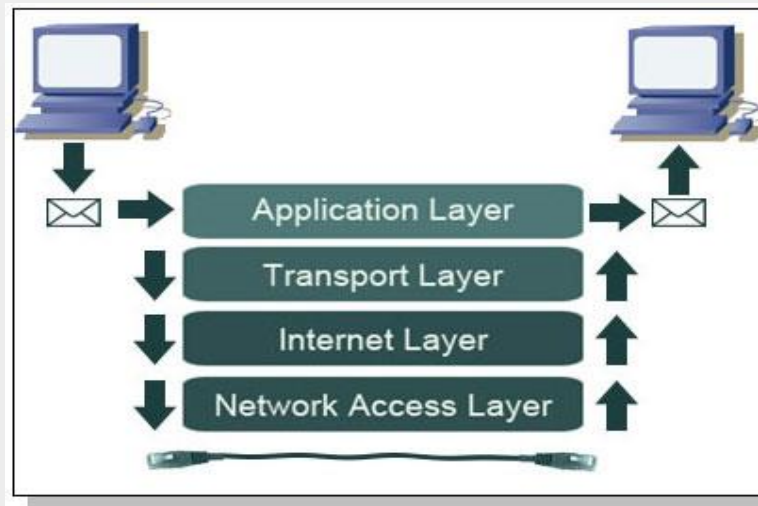
2.1. Giới thiệu

- 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.



2.1. 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.



2.1. 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*

2.1. 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*

2.1. Giới thiệu

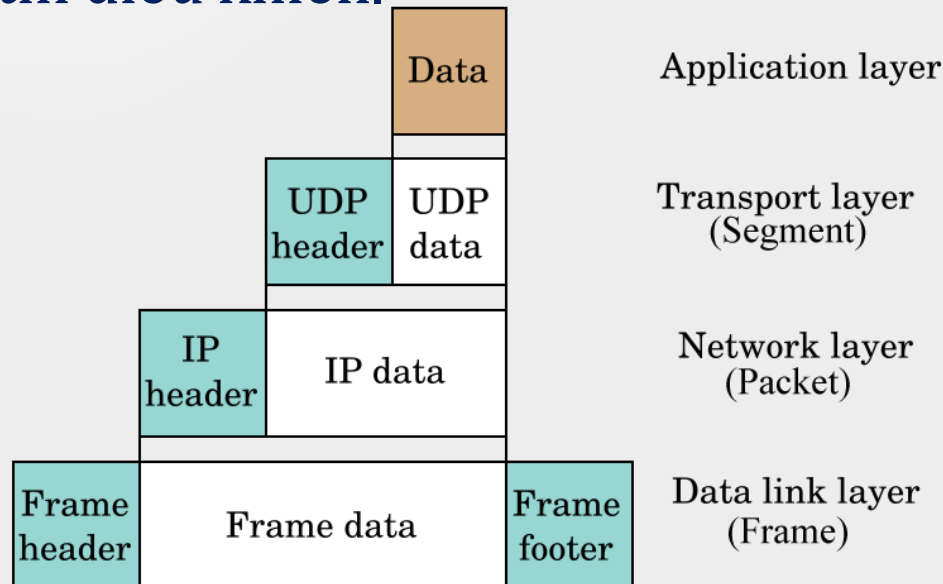
- Bộ giao thức Internet
 - Tầng Internet
 - Định tuyến và truyền các gói tin liên mạng.
 - Cung cấp dịch vụ truyền dữ liệu giữa máy tính – máy tính trong cùng nhánh mạng hoặc giữa các nhánh mạng.
 - Đơn vị dữ liệu là các gói tin (packet).
 - Các giao thức ở tầng này: IPv4, IPv6
 - *Việc lập trình ứng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

2.1. 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.*

2.1. 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.



2.1. 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



2.2. 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**

2.2. 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

2.2. 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.

2.2. 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 bit (octet).

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

2.2. Giao thức IPv4

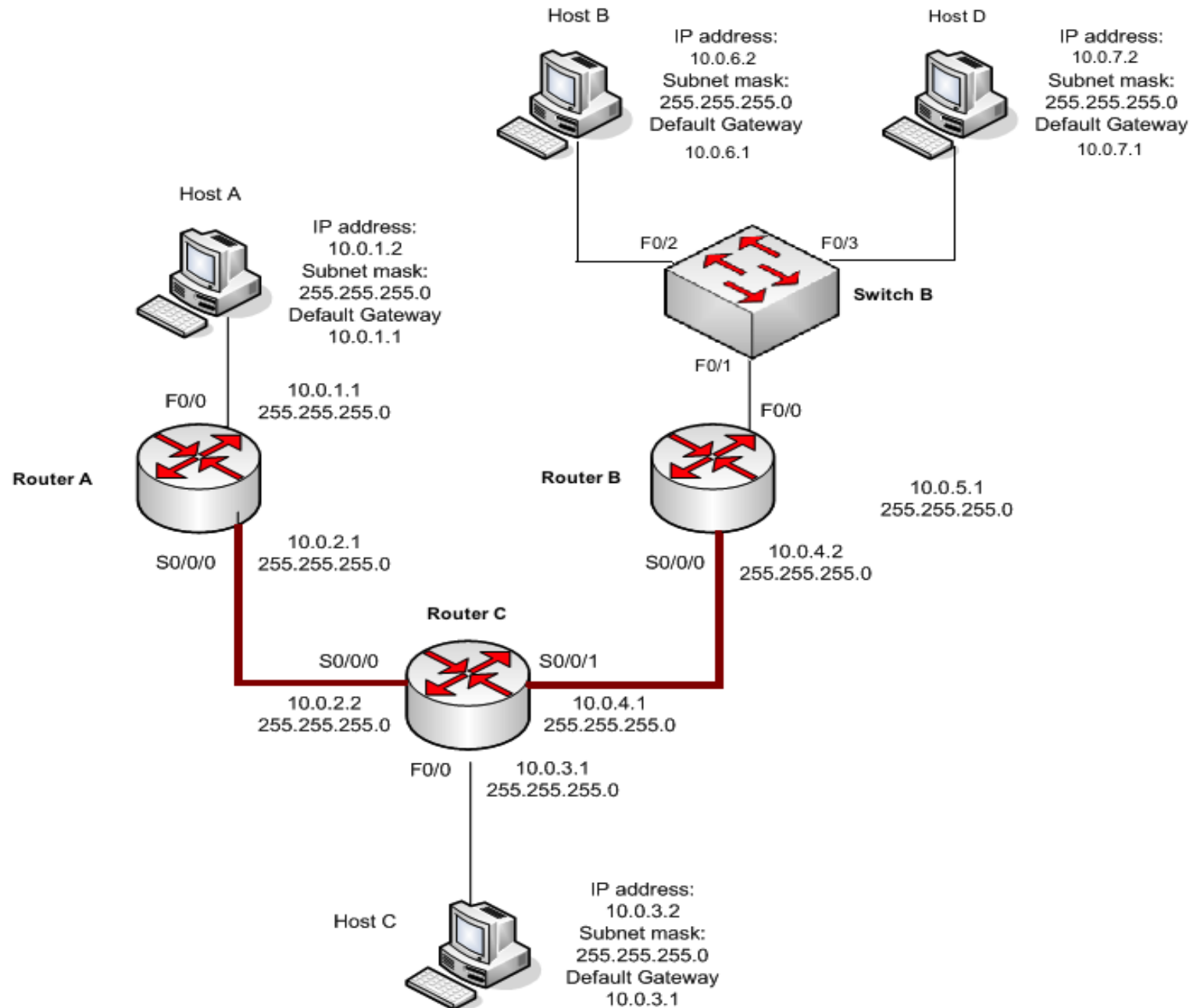
- Các lớp địa chỉ IPv4
 - Có năm lớp địa chỉ: A, B, C, D, E.
 - Lớp A, B, C: trao đổi thông tin thông thường.
 - Lớp D: **multicast**
 - Lớp E: để dành

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

2.2. 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



2.2. 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, bit 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: 0xFFFFFFFF00
 - Ít dùng

2.2. 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

2.2. 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

2.2. 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– 192.168.255.25 5	65,536	Tổ hợp từ mạng lớp C	192.168.0.0/16

2.3. Giao thức IPv6

- Giao thức IPv6
 - IETF đề xuất năm 1998.
 - Sử dụng 128 bit để đánh địa chỉ các thiết bị.
 - Khắc phục vấn đề thiếu địa chỉ của IPv4.
 - Vẫn chưa phổ biến và chưa thể thay thế hoàn toàn IPv4.

2.4. 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.*

2.4. 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)...

2.4. 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

2.4. 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.

2.4. 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 S	F Y 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

2.4. 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
```

2.5. Giao thức UDP

- Giao thức UDP: User Datagram Protocol
 - Cũng là giao thức lỗi 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...

2.5. Giao thức UDP

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

2.5. 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)

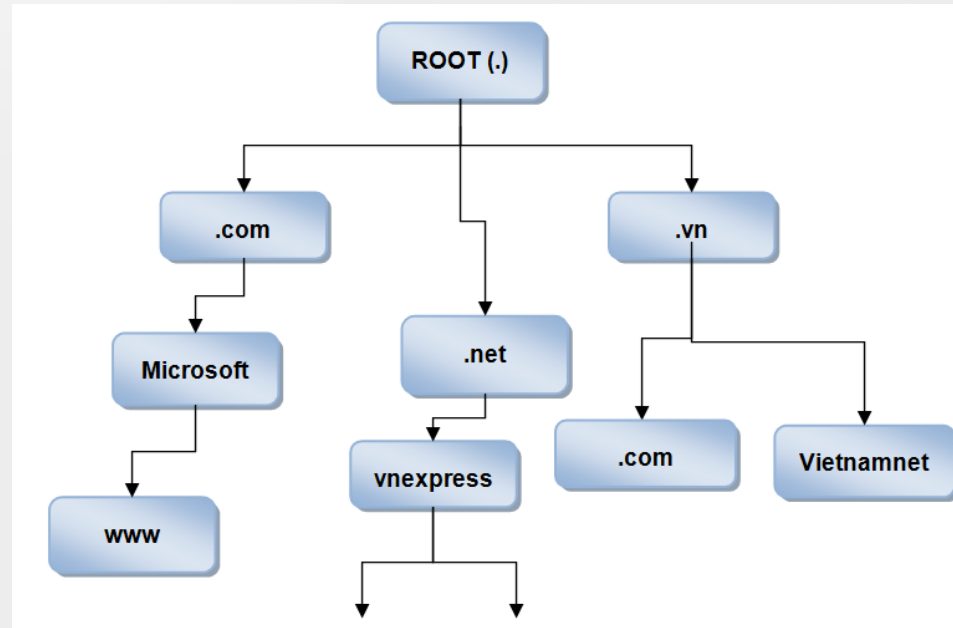
* **Checksum** được sử dụng với mục đích gì?

2.5. Giao thức UDP

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

2.6. 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.



2.6. 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

2.6. 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...

2.6. 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 3. Thư viện Windows Socket

Chương 3. Windows Socket

3.1. Kiến trúc

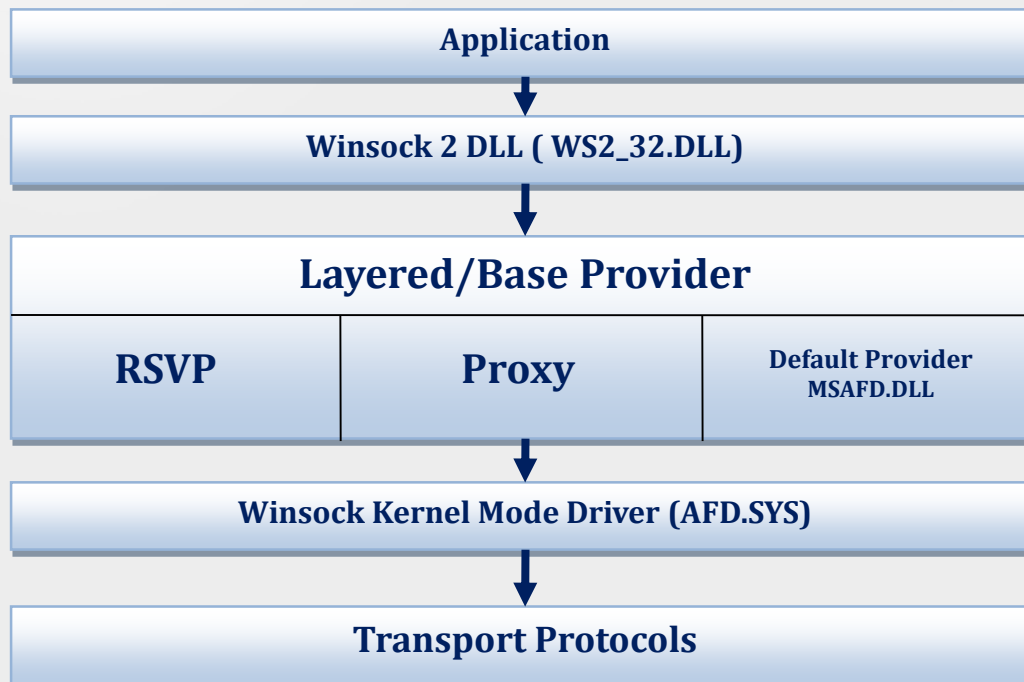
3.2. Đặc tính

3.3. Lập trình WinSock

3.4. Các phương pháp vào ra

3.1 Kiến trúc

- 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.



3.1 Kiến trúc

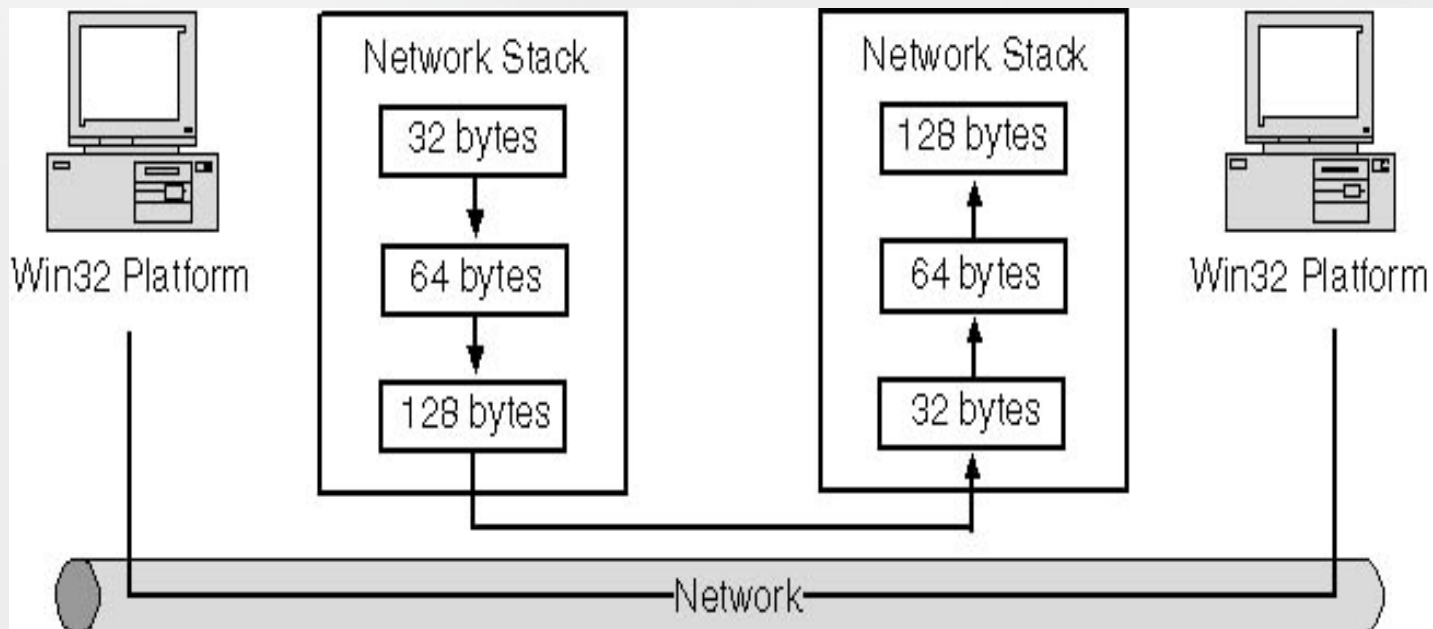
- Windows Socket (WinSock)
 - Phiên bản hiện tại là WinSock 2.2
 - Các ứng dụng sẽ giao tiếp với thư viện liên kết động ở tầng trên cùng: **WS2_32.DLL**.
 - **Provider** do nhà sản xuất của các giao thức cung cấp. Tầng này bổ sung giao thức của các tầng mạng khác nhau cho WinSock như TCP/IP, IPX/SPX, 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.

3.1 Kiến trúc

- Windows Socket (WinSock)
 - **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)
 - *Việc lập trình mạng sẽ chỉ thao tác với đối tượng **SOCKET**.*
 - *Mỗi ứng dụng cần có một **SOCKET** trước khi muốn trao đổi dữ liệu với ứng dụng khác.*
 - *Đường dây ảo nối giữa các **SOCKET** sẽ là kênh truyền dữ liệu của hai ứng dụng.*

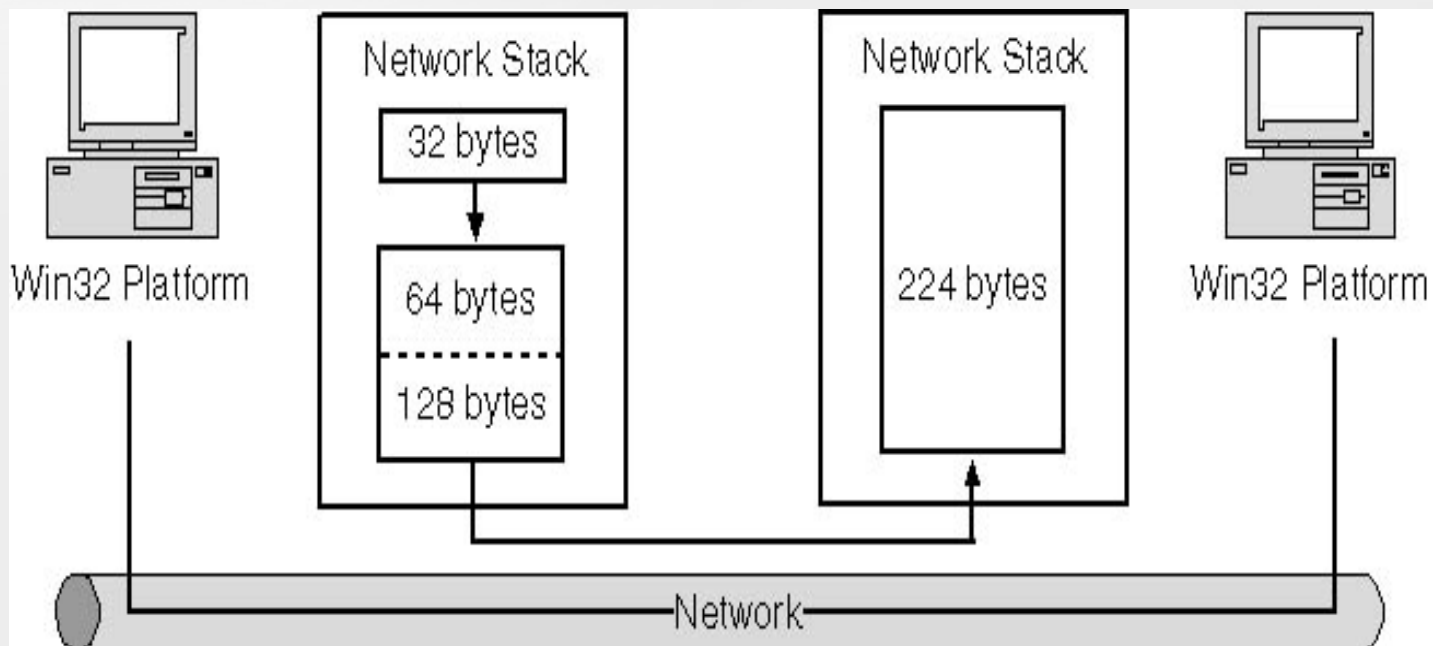
3.2 Đặc tính

- 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



3.2 Đặc tính

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



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. Ví dụ: TCP
 - Giao thức không kết nối (connectionless) không cần thiết lập kênh truyền trước khi truyền. Ví dụ: UDP

3.2 Đặc tính

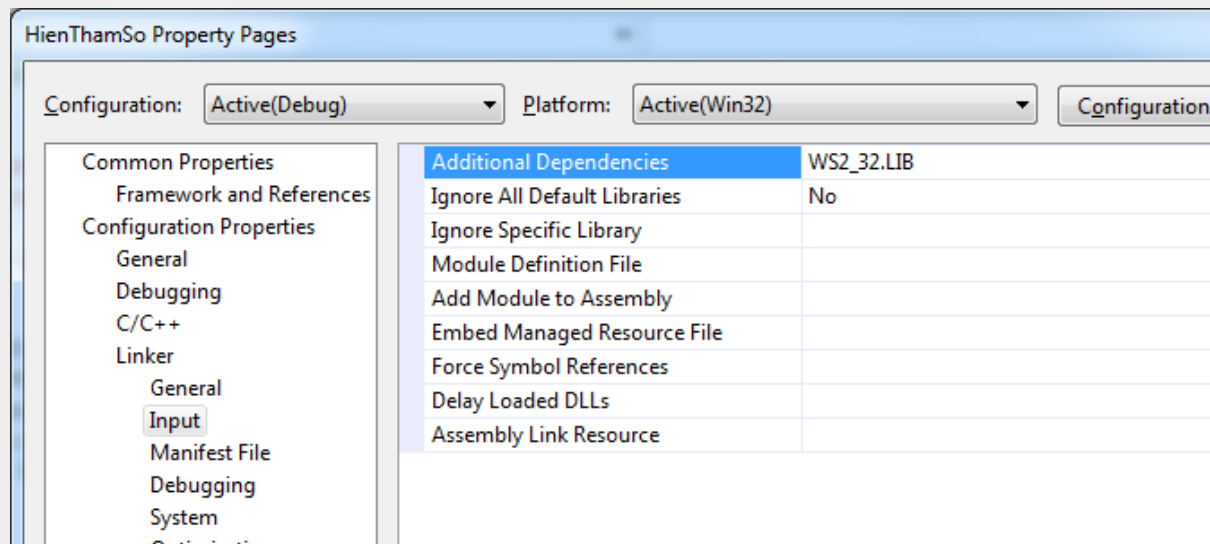
- Hỗ trợ các giao thức tin cậy và trật tự
 - Tin cậy (reliability): đảm bảo chính xác từng byte được gửi đến đích.
 - Trật tự (ordering): đảm bảo chính xác trật tự từng byte dữ liệu. Byte nào gửi trước sẽ được nhận trước, byte gửi sau sẽ được nhận sau.

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.

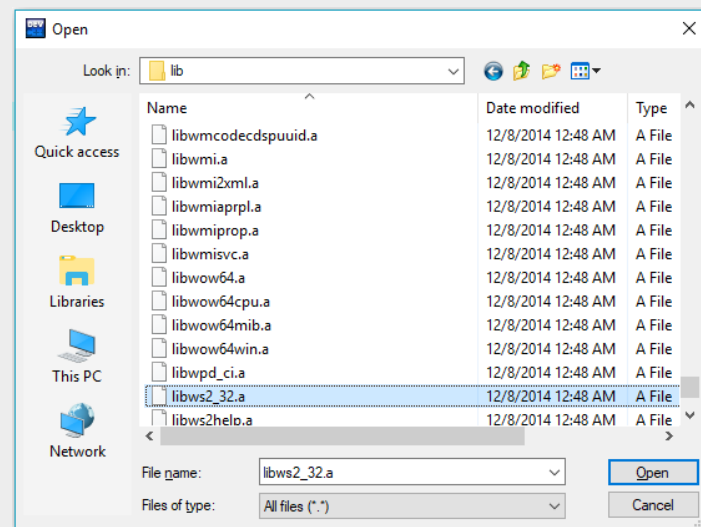
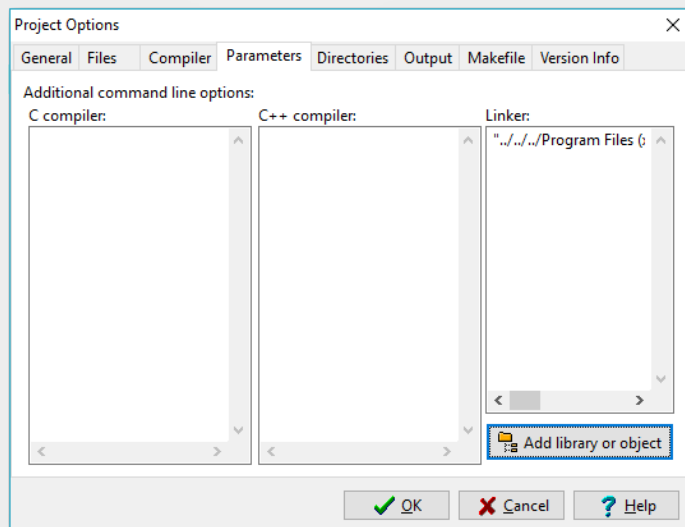
3.3 Lập trình WinSock

- Chuẩn bị môi trường:
 - Hệ điều hành Windows XP/7/8/10.
 - 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



3.3 Lập trình WinSock

- Chuẩn bị môi trường:
 - Hệ điều hành Windows XP/7/8/10.
 - 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**



3.3 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

3.3 Lập trình WinSock

- Khởi tạo WinSock
 - Ví dụ:

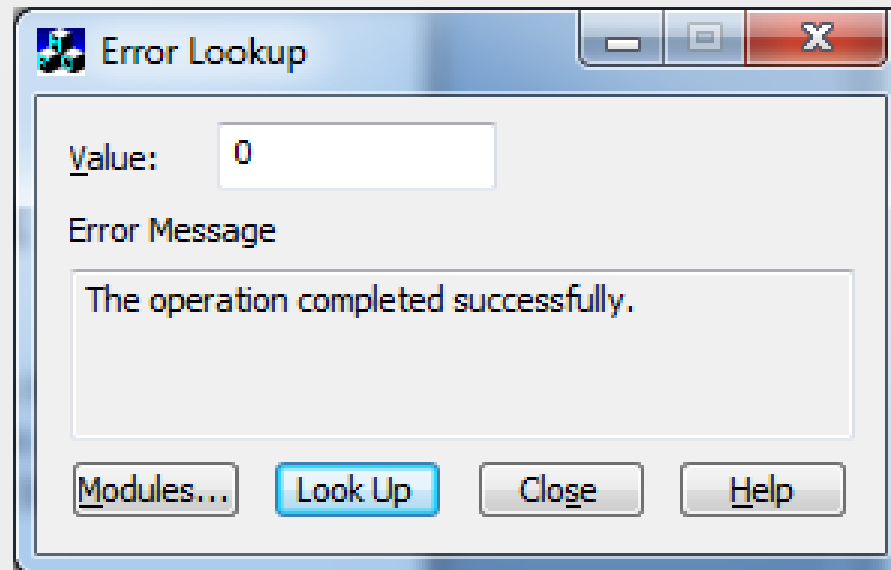
```
WSADATA wsaData;  
WORD wVersion = MAKEWORD(2, 2); // Khởi tạo phiên bản 2.2  
if (WSAStartup(wVersion, &wsaData)) {  
    printf("Version not supported");  
}
```

3.3 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

3.3 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)



3.3 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

3.3 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);
```

3.3 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  
};
```

3.3 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)`

3.3 Lập trình WinSock

- Xác định địa chỉ
 - Ví dụ: g địa chỉ 192.168.0.1:80 vào cấu trúc sockaddr_in

```
SOCKADDR_IN InternetAddr; // Khai báo biến lưu địa chỉ

InternetAddr.sin_family = AF_INET; // Họ địa chỉ Internet

//Chuyển xâu địa chỉ 192.168.0.1 sang số 4 byte dạng
// network-byte order và gán cho trường sin_addr
InternetAddr.sin_addr.s_addr = inet_addr("192.168.0.1");

//Chuyển đổi cổng sang dạng network-byte order
// và gán cho trường sin_port
InternetAddr.sin_port = htons(80);
```

3.3 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()**

3.3 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 thức 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  
};
```


3.3 Lập trình WinSock

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

```
addrinfo * result;    // Lưu kết quả phân giải
int rc;               // Lưu mã trả về
sockaddr_in address;  // Lưu địa chỉ phân giải được
rc = getaddrinfo("www.hust.edu.vn", "http", NULL, &result);

// Một tên miền có thể có nhiều địa chỉ IP tương ứng
// Lấy kết quả đầu tiên
if (rc == 0)
    memcpy(&address, result->ai_addr, result->ai_addrlen);

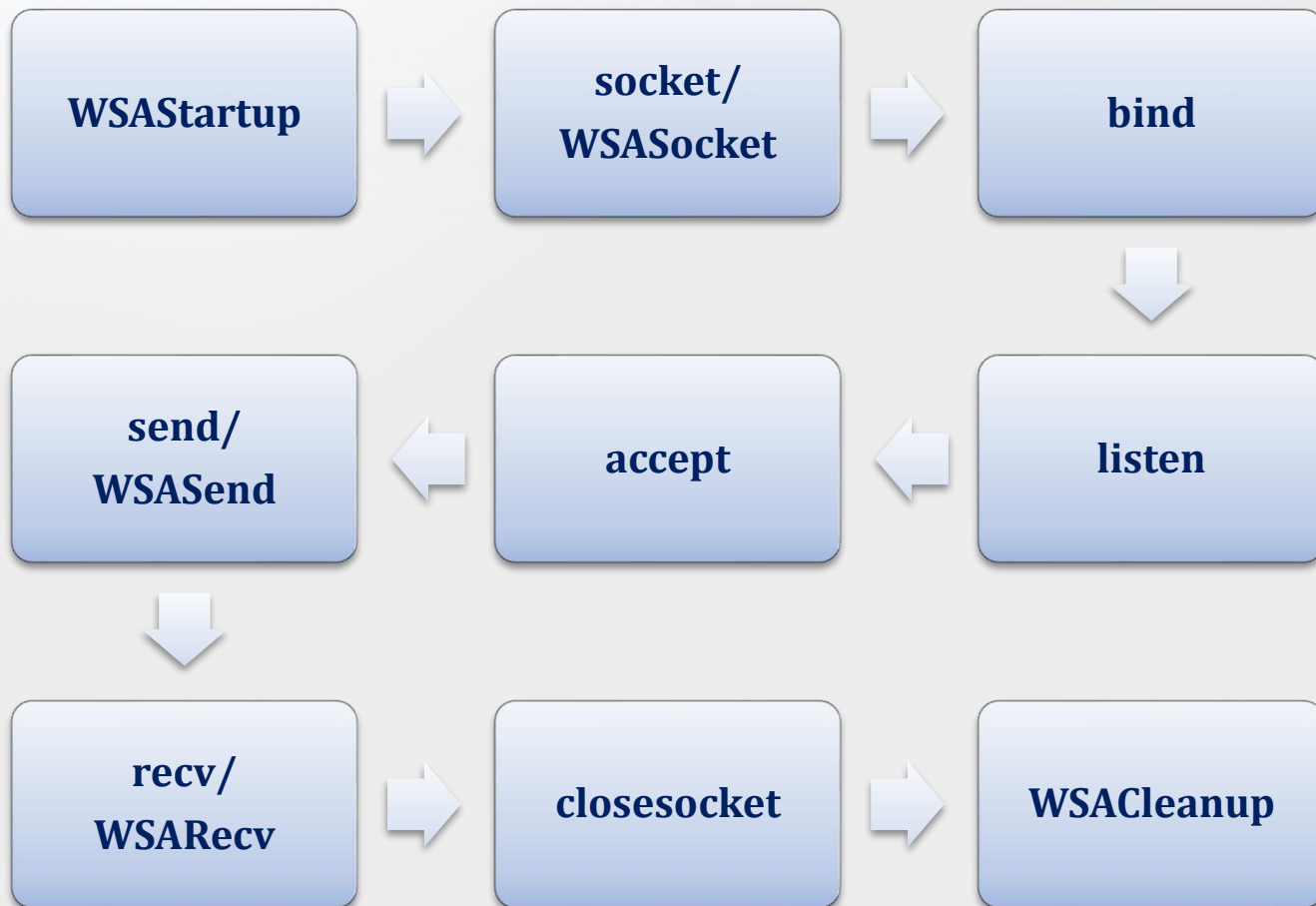
// Xử lý với address...
```

3.3 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**

3.3 Lập trình WinSock

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



3.3 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:

```
SOCKADDR_IN tcpaddr;
short port = 8888;
tcpaddr.sin_family = AF_INET; // Socket IPv4
tcpaddr.sin_port = htons(port); // host order => net order
tcpaddr.sin_addr.s_addr = htonl(INADDR_ANY); //Giao diện bất kỳ
bind(s, (SOCKADDR *)&tcpaddr, sizeof(tcpaddr)); // Bind socket
```

3.3 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

3.3 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.

3.3 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
```

3.3 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

3.3 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;
send(client, &l, sizeof(l), 0);
```

3.3 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

3.3 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
}
```

3.3 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

3.3 Lập trình WinSock

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

```
#include <winsock2.h>                                //Thu vien Winsock
void main(void)
{
    WSADATA          wsaData;
    SOCKET           ListeningSocket;
    SOCKET           NewConnection;
    SOCKADDR_IN      ServerAddr;
    SOCKADDR_IN      ClientAddr;
    int              ClientAddrLen;
    int              Port = 8888;
    // Khoi tao Winsock 2.2
    WSStartup(MAKEWORD(2,2), &wsaData);
    // Tao socket lang nghe ket noi tu client.
    ListeningSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    // Khoi tao cau truc SOCKADDR_IN cua server
    // doi ket noi o cong 8888
    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_port = htons(Port);
    ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

3.3 Lập trình WinSock

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

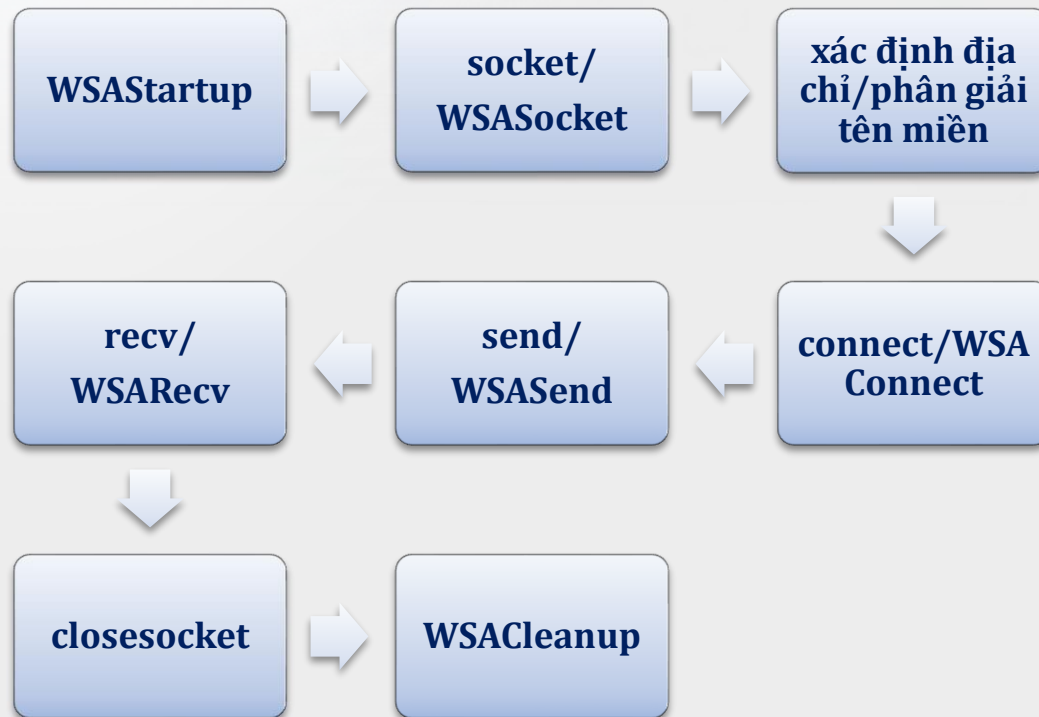
```
// Bind socket của server.
bind(ListeningSocket, (SOCKADDR *)&ServerAddr,
    sizeof(ServerAddr));
// Chuyển sang trạng thái doi ket noi
listen(ListeningSocket, 5);
// Chấp nhận ket noi moi.
ClientAddrLen = sizeof(ClientAddr);
NewConnection = accept(ListeningSocket, (SOCKADDR *)
    &ClientAddr, &ClientAddrLen);
// Sau khi chấp nhận ket noi, server có thể tiếp tục chấp nhận
// thêm các ket noi khác, hoặc gửi nhận dữ liệu với các client thông
// qua các socket được accept với client
// Đóng socket
closesocket(NewConnection);
closesocket(ListeningSocket);
// Giải phóng Winsock
WSACleanup();
}
```

3.3 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**

3.3 Lập trình WinSock

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



3.3 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

3.3 Lập trình WinSock

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

```
#include <winsock2.h>
void main(void)
{
    WSADATA    wsaData;
    SOCKET     s;
    SOCKADDR_IN ServerAddr;
    int        Port = 8888;
    // Khởi tạo Winsock 2.2
    WSStartup(MAKEWORD(2,2), &wsaData);
    // Tạo socket client
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    // Khởi tạo cấu trúc SOCKADDR_IN có địa chỉ server là
    202.191.56.69 và cổng 8888

    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_port = htons(Port);
    ServerAddr.sin_addr.s_addr = inet_addr("202.191.56.69");
```

3.3 Lập trình WinSock

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

```
// Ket noi den server thong qua socket s.  
connect(s, (SOCKADDR *) &ServerAddr, sizeof(ServerAddr));  
  
// Bat dau gui nhan du lieu  
// Su dung cac lenh send, recv de truyen, nhan du lieu  
  
// Ket thuc gui nhan du lieu  
// Dong socket  
closesocket(s);  
  
// Giai phong Winsock  
WSACleanup();  
}
```

3.3 Lập trình WinSock

- Ví dụ minh họa
 - Tạo client gửi thông điệp đến netcat server
 - Tạo client gửi lệnh GET đến news.zing.vn và hiển thị kết quả trả về
 - 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

3.3 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



3.3 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

3.3 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 addr; // 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 *)& addr,
sizeof(addr));
```

3.3 Lập trình WinSock

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



3.3 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

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
 - Đoạn chương trình sau sẽ nhận dữ 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));
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
 - Đoạn chương trình (tiếp)

```
// Lắp 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); // Hiện thị ra màn hình
    }
}
```

3.3 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 <công 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 <công 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`**

3.3 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 câu 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

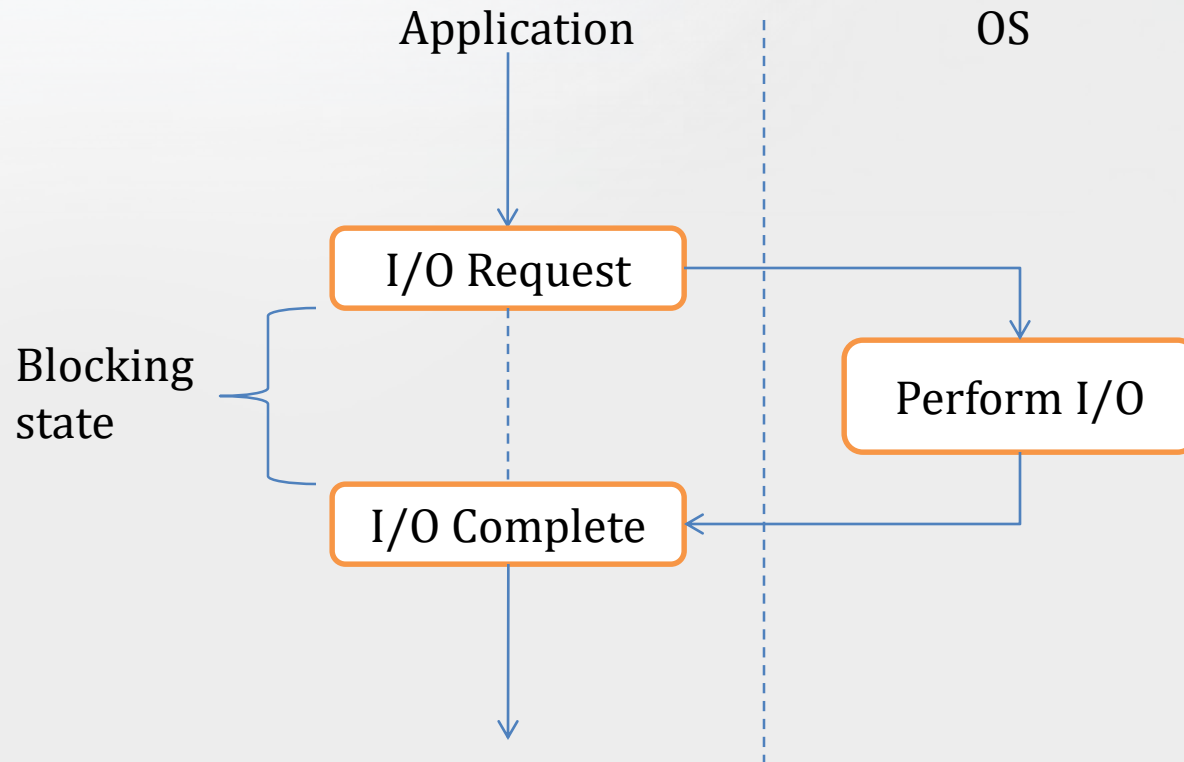
VD: `serverinfo.exe 1234`

3.4 Các phương pháp vào ra

- Thread (Luồng):
 - Là đơn vị thực thi độc lập và tuần tự của chương trình.
 - Mỗi chương trình có ít nhất một thread chính là thread bắt đầu thực hiện tại hàm **main**
- 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**
 - ...

3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
 - Blocking (Đồng bộ):



3.4 Các phương pháp vào ra

- 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

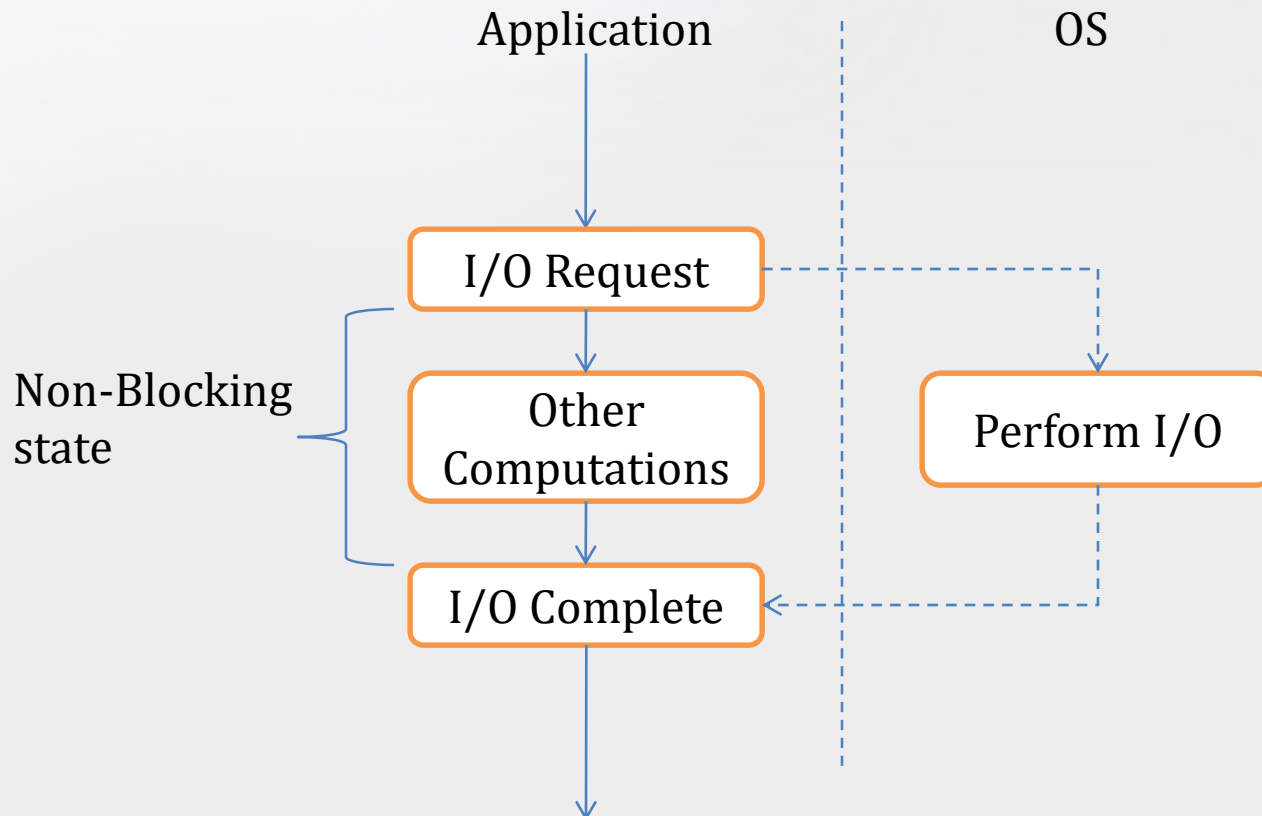
```
...  
do {  
    // Thread sẽ bị chặn lại khi gọi hàm recvfrom  
    // Trong lúc đợi dữ liệu thì không thể gửi dữ liệu  
    rc = recvfrom(receiver, szXau, 128, 0,  
                  (sockaddr*)&senderAddress, &senderLen);  
    // ...  
} while ()  
...
```

3.4 Các phương pháp vào ra

- 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.

3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
 - Non-Blocking (Bất đồng bộ):



3.4 Các phương pháp vào ra

- 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  
}
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
 - Mô hình Blocking – Multithread
 - Mô hình Select
 - Mô hình WSAAsyncSelect
 - Mô hình WSAEventSelect
 - Mô hình Overlapped IO
 - Cơ chế Event Notification
 - Cơ chế Completion Routine
 - Mô hình Completion Port

3.4 Các phương pháp vào ra

- Mô hình **Blocking**

- 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

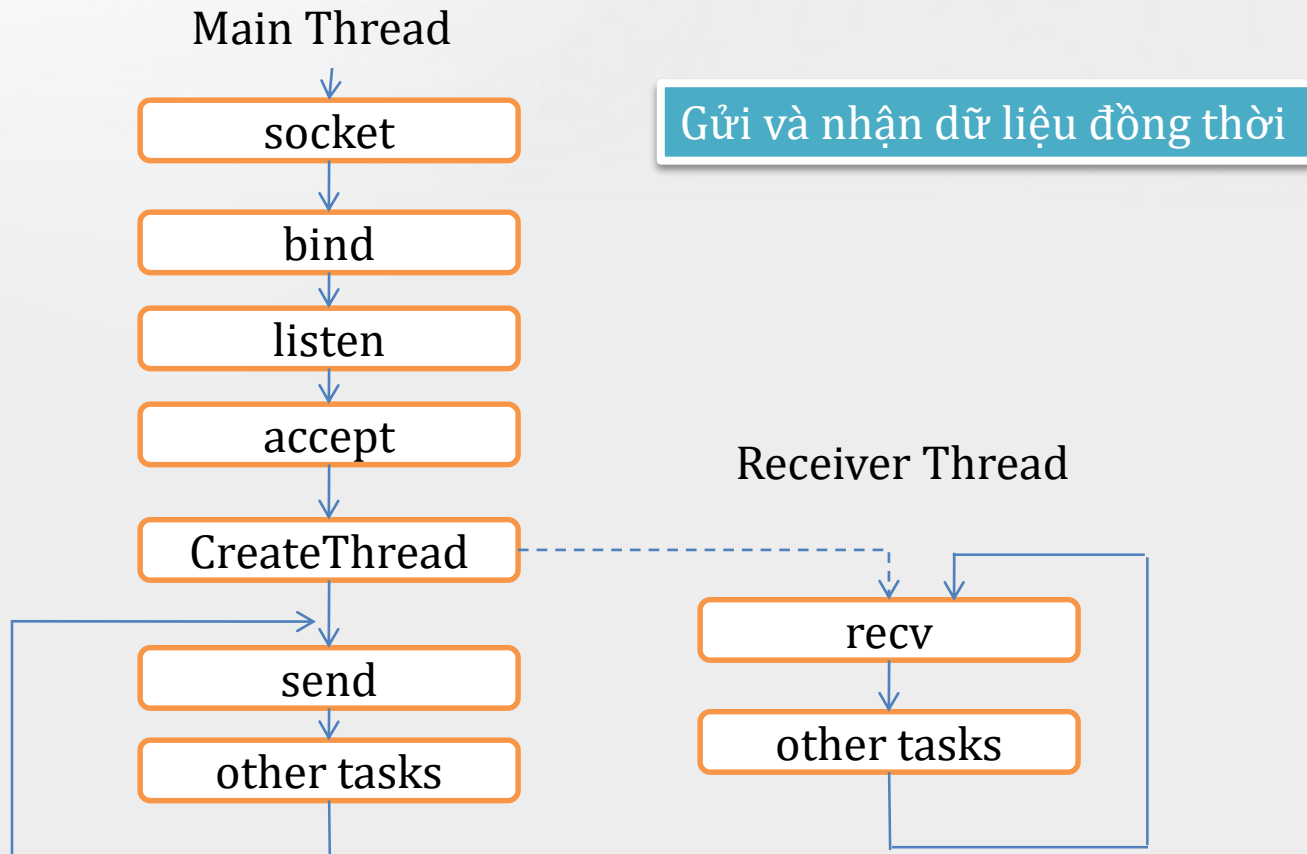
```
HANDLE WINAPI CreateThread(  
    __in LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    __in SIZE_T dwStackSize,  
    __in LPTHREAD_START_ROUTINE lpStartAddress,  
    __in LPVOID lpParameter,  
    __in DWORD dwCreationFlags,  
    __out LPDWORD lpThreadId );
```

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

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

3.4 Các phương pháp vào ra

- Mô hình **Blocking**



3.4 Các phương pháp vào ra

- Mô hình **Blocking**
 - 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);
}
...
```

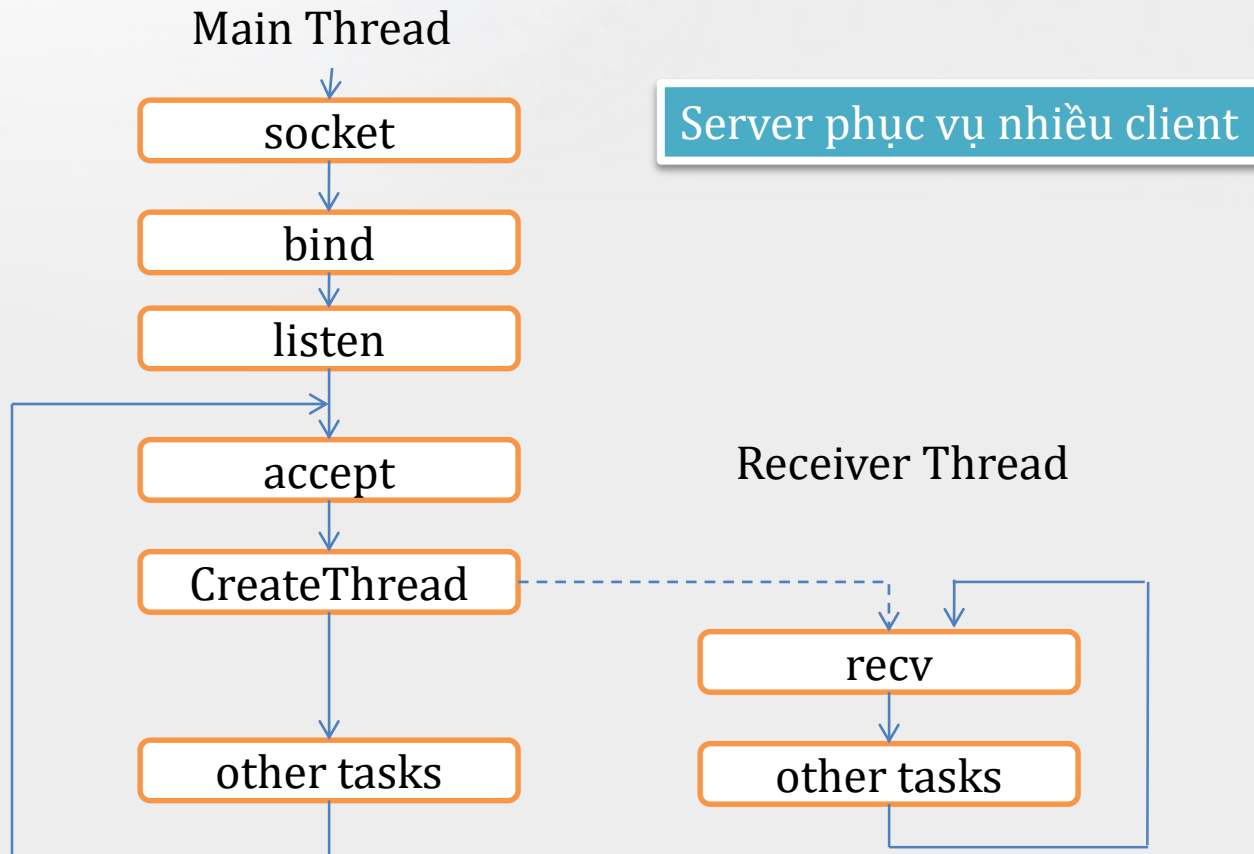
3.4 Các phương pháp vào ra

- Mô hình **Blocking**
 - 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);
}
```

3.4 Các phương pháp vào ra

- Mô hình **Blocking**



Bài tập: Chat server

Viết chương trình chatroom server nhận tham số cổng từ dòng lệnh, nghe và phục vụ các client làm 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 xxxxxxxx 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 chào" hoặc có thể thêm thời gian vào trước ví dụ: "2014/05/06 11:00:00PM abc: xin chào"

Bài tập: Telnet server

Viết một chương trình telnet server (mytelnets) làm nhiệm vụ sau:

- Đợi kết nối ở cổng được truyền vào từ tham số dòng lệnh, ví dụ: “mytelnets 5000” thì đợi ở cổng 5000.
- 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
- Sử dụng Thread để server có thể xử lý nhiều thao tác, nhiều kết nối.

3.4 Các phương pháp vào ra

- Mô hình **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).

3.4 Các phương pháp vào ra

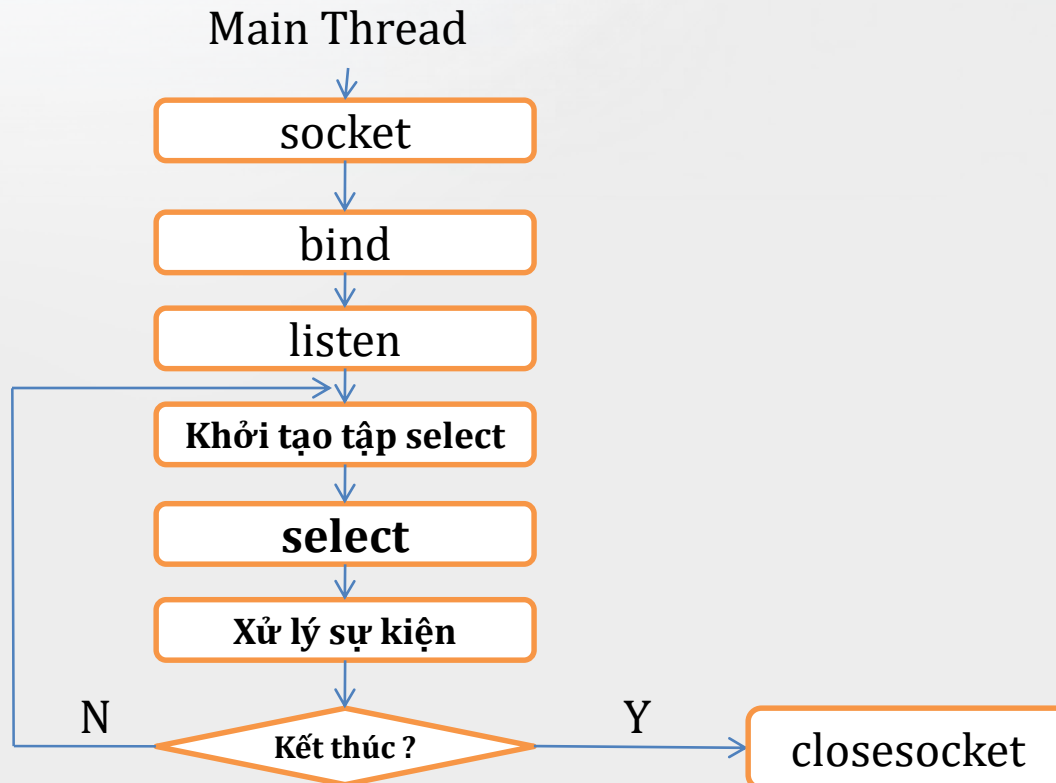
- Mô hình **select**
 - Nguyên mẫu hàm như sau

```
int select(  
    int nfd, // 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

3.4 Các phương pháp vào ra

- Mô hình **select**



3.4 Các phương pháp vào ra

- 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.

3.4 Các phương pháp vào ra

- 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  
    }  
}
```

3.4 Các phương pháp vào ra

- 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  
    }  
}  
}
```

3.4 Các phương pháp vào ra

- 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)

3.4 Các phương pháp vào ra

- 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  
);
```

3.4 Các phương pháp vào ra

- 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 sổ:

```
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

3.4 Các phương pháp vào ra

- 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;  
            }  
    }
```


3.4 Các phương pháp vào ra

- 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;
}
```

3.4 Các phương pháp vào ra

- 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;
```

3.4 Các phương pháp vào ra

- 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);
}
```

3.4 Các phương pháp vào ra

- 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.

Bài tập

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

3.4 Các phương pháp vào ra

- 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) ;

3.4 Các phương pháp vào ra

- 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_WRITE);
```

3.4 Các phương pháp vào ra

- 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.

3.4 Các phương pháp vào ra

- 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_OBJECT_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

3.4 Các phương pháp vào ra

- 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,
                                       1000, FALSE);
        if ((ret != WSA_WAIT_FAILED) && (ret != WSA_WAIT_TIMEOUT))
        {
            // Service event signaled on events[index]
        }
    }
}
```

3.4 Các phương pháp vào ra

- 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;
```

3.4 Các phương pháp vào ra

- 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
}
else 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
}
```

3.4 Các phương pháp vào ra

- Mô hình **WSAEventSelect**
 - Ví dụ

```
#include <winsock2.h>
#define MAX_EVENTS 64
int _tmain(int argc, _TCHAR* argv[]) {
    SOCKET SocketArray [MAX_EVENTS];
    WSAEVENT EventArray [MAX_EVENTS], NewEvent;
    SOCKADDR_IN InternetAddr;
    SOCKET Accept, Listen;
    DWORD EventTotal = 0;
    DWORD Index, i;
    WSADATA wsaData;
    WORD wVersion = MAKEWORD(2, 2);
    int rc = WSAStartup(wVersion, &wsaData);
    // Thiết lập TCP socket đợi kết nối ở 8888
    Listen = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
    InternetAddr.sin_family = AF_INET;
    InternetAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    InternetAddr.sin_port = htons(8888);
    rc = bind(Listen, (PSOCKADDR) &InternetAddr, sizeof(InternetAddr));
```

3.4 Các phương pháp vào ra

- Mô hình **WSAEventSelect**

- Ví dụ

```
SOCKET Accept, Listen;
...
NewEvent = WSACreateEvent();
WSAEventSelect(Listen, NewEvent, FD_ACCEPT | FD_CLOSE);
rc = listen(Listen, 5);
WSANETWORKEVENTS NetworkEvents;
SocketArray[EventTotal] = Listen;
EventArray[EventTotal] = NewEvent;
EventTotal++;
char buffer[1024];
int len;
while(TRUE)
{
    // Đợi tất cả các sự kiện
    Index = WSAWaitForMultipleEvents(EventTotal, EventArray, FALSE,
                                     WSA_INFINITE, FALSE);
    Index = Index - WSA_WAIT_EVENT_0;
```

3.4 Các phương pháp vào ra

- Mô hình **WSAEventSelect**
 - Ví dụ

```
// Duyệt để tìm ra sự kiện nào được báo hiệu
for (i=Index; i < EventTotal ;i++) {
    Index = WSAWaitForMultipleEvents(1, &EventArray[i], TRUE,
    1000, FALSE);
    if (Index == WSA_WAIT_FAILED || Index == WSA_WAIT_TIMEOUT)
        continue;
    else {
        Index = i;
        WSAResetEvent(EventArray[Index]);
        WSAEnumNetworkEvents(SocketArray[Index], EventArray[Index],
            &NetworkEvents);
    }
}
```

3.4 Các phương pháp vào ra

- Mô hình **WSAEventSelect**
 - Ví dụ

```
// Kiểm tra sự kiện FD_ACCEPT
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]);
        break;
    }

    // Chấp nhận kết nối mới
    // cho vào danh sách socket và sự kiện
    Accept = accept(SocketArray[Index], NULL, NULL);
```


3.4 Các phương pháp vào ra

- Mô hình **WSAEventSelect**

- Ví dụ

```
    if (EventTotal > WSA_MAXIMUM_WAIT_EVENTS) {
        printf("Too many connections");
        closesocket(Accept);
        break;
    }

    NewEvent = WSACreateEvent();

    WSAEventSelect(Accept, NewEvent, FD_READ|FD_WRITE|FD_CLOSE);

    EventArray[EventTotal] = NewEvent;
    SocketArray[EventTotal] = Accept;
    EventTotal++;
    printf("Socket %d connected\n", Accept);
}
```

...

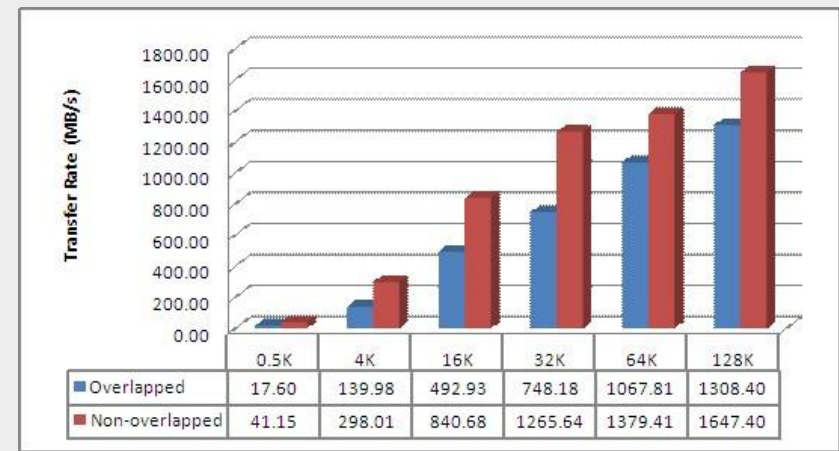
Bài tập

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

3.4 Các phương pháp vào ra

- 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:
 - WSA_send
 - WSA_sendto
 - **WSARecv**
 - WSARecvFrom
 - WSAIoctl
 - WSARecvMsg
 - AcceptEx
 - ConnectEx
 - TransmitFile
 - TransmitPackets
 - DisconnectEx
 - WSANSPIoctl



3.4 Các phương pháp vào ra

- 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**

3.4 Các phương pháp vào ra

- 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

3.4 Các phương pháp vào ra

- 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

3.4 Các phương pháp vào ra

- 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.

3.4 Các phương pháp vào ra

- 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**

3.4 Các phương pháp vào ra

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

```
// Khởi tạo WinSock và kết nối đến 127.0.0.1:8888
...
OVERLAPPED    overlapped; // Khai báo cấu trúc OVERLAPPED
WSAEVENT receiveEvent = WSACreateEvent(); // Tạo event
memset(&overlapped,0,sizeof(overlapped));
overlapped.hEvent = receiveEvent;

charbuff[1024]; // Bộ đệm nhận dữ liệu
WSABUF  databuff; // Cấu trúc mô tả bộ đệm
databuff.buf = buff;
databuff.len = 1024;
DWORD   bytesReceived = 0; // Số byte nhận được
DWORD   flags = 0; // Cờ quy định cách nhận, bắt buộc phải có

while (1)
{
    // Gửi yêu cầu nhận dữ liệu
    rc = WSARecv(s,&databuff,1,&bytesReceived,&flags,&overlapped,0);
```

3.4 Các phương pháp vào ra

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

```
if (rc == SOCKET_ERROR) {
    rc = WSAGetLastError();
    if (rc != WSA_IO_PENDING) {
        printf("Loi %d !\n", rc);
        continue;
    }
}
rc = WSAWaitForMultipleEvents(1, &receiveEvent, TRUE, WSA_INFINITE, FALSE);
if ((rc == WSA_WAIT_FAILED) || (rc == WSA_WAIT_TIMEOUT)) continue;
WSAResetEvent(receiveEvent);
rc = WSAGetOverlappedResult(s, &overlapped, &bytesReceived, FALSE, &flags);
if (bytesReceived == 0)
    break;
// Hiển thị
buff[bytesReceived] = 0;
printf(buff);
}
```

3.4 Các phương pháp vào ra

- 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.

3.4 Các phương pháp vào ra

- 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
```

3.4 Các phương pháp vào ra

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

```
// Khai báo các cấu trúc cần thiết
SOCKET s;
OVERLAPPED overlapped;
char buff[1024];
WSABUF databuff;
DWORD flags;
DWORD bytesReceived = 0;
int rc = 0;

void CALLBACK CompletionRoutine(DWORD dwError, DWORD dwTransferred,
LPWSAOVERLAPPED lpOverlapped, DWORD dwFlags) {
    if (dwError != 0 || dwTransferred == 0) { // Xử lý lỗi
        closesocket(s);
        return;
    }
};
```

3.4 Các phương pháp vào ra

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

```
// Hiển thị xâu ra màn hình
buff[dwTransferred] = 0;
printf(buff);
// Khởi tạo lại cấu trúc overlapped và lại gửi tiếp yêu cầu nhận
dữ liệu
memset(&overlapped, 0, sizeof(overlapped));
flags = 0;
rc = WSARecv(s, &databuff, 1, &bytesReceived, &flags,
&overlapped, CompletionRoutine);
if (rc == SOCKET_ERROR) {
    rc = WSAGetLastError();
    if (rc != WSA_IO_PENDING)
        printf("Loi %d !\n",rc);
};
return;
}
```

3.4 Các phương pháp vào ra

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

```
int _tmain(int argc, _TCHAR* argv){
    // Khởi tạo và kết nối đến 127.0.0.1:8888
    ...
    // Khởi tạo cấu trúc overlapped
    memset(&overlapped,0,sizeof(overlapped));
    // Khởi tạo bộ đệm dữ liệu
    databuff.buf = buff;
    databuff.len = 1024;
    // Gửi yêu cầu vào ra
    rc = WSARecv(s, &databuff, 1, &bytesReceived, &flags,
    &overlapped, CompletionRoutine);
    // Xử lý lỗi...
    // Chuyển luồng sang trạng thái alertable
    while (1) SleepEx(1000,TRUE);

    closesocket(s);
    WSACleanup();
    return 0;
}
```

3.4 Các phương pháp vào ra

- 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

3.4 Các phương pháp vào ra

- 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 socket 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

3.4 Các phương pháp vào ra

- 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);
```

3.4 Các phương pháp vào ra

- 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);  
}
```

3.4 Các phương pháp vào ra

- 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)

3.4 Các phương pháp vào ra

- 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,
    (DWORD)PerHandleData, 0);
```

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

3.4 Các phương pháp vào ra

- 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

3.4 Các phương pháp vào ra

- 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,
(LPDWORD)&PerHandleData, (LPOVERLAPPED *)&PerIoData, INFINITE);
```

3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
#include "stdafx.h"
#include "winsock2.h"

// Khai báo cấu trúc dữ liệu socket
typedef struct _PER_HANDLE_DATA
{
    SOCKET Socket;
    SOCKADDR_STORAGE ClientAddr;
} PER_HANDLE_DATA, * LPPER_HANDLE_DATA;

// Khai báo cấu trúc dữ liệu overlapped
typedef struct _PER_IO_DATA
{
    OVERLAPPED Overlapped;
    WSABUF DataBuf;
    char buf[1024];
} PER_IO_DATA, * LPPER_IO_DATA;

// Nguyên mẫu hàm thread
DWORD WINAPI ServerWorkerThread(LPVOID);
```


3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
int main()
{
HANDLE CompletionPort;
WSADATA wsa;
SYSTEM_INFO SystemInfo;
SOCKADDR_IN InternetAddr;
SOCKET Listen;
int i;

PER_HANDLE_DATA *PerHandleData = NULL;
PER_IO_DATA *PerIoData = NULL;
DWORD Bytes, Flags;

WSAStartup(MAKEWORD(2, 2), &wsa);

// Tạo đối tượng CompletionPort
CompletionPort = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
```

3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
// Lấy số lượng processor của hệ thống
GetSystemInfo(&SystemInfo);

// Tạo các worker thread tương ứng với số lượng processor
for (i = 0; i < SystemInfo.dwNumberOfProcessors; i++)
{
    HANDLE ThreadHandle;
    ThreadHandle = CreateThread(NULL, 0, ServerWorkerThread, CompletionPort,
0, NULL);
    CloseHandle(ThreadHandle);
}

Listen = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

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

bind(Listen, (SOCKADDR *)&InternetAddr, sizeof(InternetAddr));

listen(Listen, 5);
```

3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
while (1)
{
    SOCKADDR_IN saRemote;
    SOCKET Accept;
    int RemoteLen;
    RemoteLen = sizeof(saRemote);

    // Chấp nhận kết nối đến server
    Accept = accept(Listen, (SOCKADDR *)&saRemote, &RemoteLen);

    // Cấp phát bộ nhớ cho cấu trúc dữ liệu socket
    PerHandleData = (LPPER_HANDLE_DATA)GlobalAlloc(GPTR,
    sizeof(PER_HANDLE_DATA));

    // Lưu thông tin socket vào cấu trúc dữ liệu
    printf("Socket number %d connected\n", Accept);
    PerHandleData->Socket = Accept;
    memcpy(&PerHandleData->ClientAddr, &saRemote, RemoteLen);
}
```

3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
// Gắn socket với đối tượng CompletionPort
CreateIoCompletionPort((HANDLE)Accept, CompletionPort,
    (DWORD)PerHandleData, 0);

Bytes = 0;
Flags = 0;

// Cấp phát bộ nhớ cho cấu trúc dữ liệu Overlapped IO
PerIoData = (LPPER_IO_DATA)GlobalAlloc(GPTR, sizeof(PER_IO_DATA));

PerIoData->DataBuf.len = 1024;
PerIoData->DataBuf.buf = PerIoData->buf;

// Yêu cầu nhận dữ liệu lần đầu
WSARecv(Accept, &(PerIoData->DataBuf), 1, &Bytes, &Flags,
    &(PerIoData->Overlapped), NULL);
}

return 0;
```

3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
// Thread xử lý việc chờ các yêu cầu vào ra và nhận dữ liệu
DWORD WINAPI ServerWorkerThread(LPVOID CompletionPortID)
{
    HANDLE CompletionPort = (HANDLE)CompletionPortID;
    DWORD BytesTransferred;
    LPPER_HANDLE_DATA PerHandleData;
    LPPER_IO_DATA PerIoData;
    DWORD Flags;

    while (TRUE)
    {
        // Chờ đến khi yêu cầu vào ra hoàn tất
        bool ret = GetQueuedCompletionStatus(CompletionPort, &BytesTransferred,
            (LPDWORD)&PerHandleData, (LPOVERLAPPED *)&PerIoData, INFINITE);
        if (BytesTransferred == 0) // Nếu truyền nhận bị lỗi
        {
            closesocket(PerHandleData->Socket); // Ngắt kết nối
            GlobalFree(PerHandleData);           // và giải phóng bộ nhớ
            GlobalFree(PerIoData);
            continue;
        }
    }
}
```

3.4 Các phương pháp vào ra

- Mô hình **Completion Port** – Ví dụ

```
// Xử lý dữ liệu nhận được
PerIoData->buf[BytesTransferred] = 0;
printf("Received Data: %s\n", PerIoData->buf);

// Khởi tạo lại cấu trúc Overlapped
Flags = 0;
ZeroMemory(&(PerIoData->Overlapped), sizeof(OVERLAPPED));

// Yêu cầu nhận dữ liệu tiếp theo
WSARecv(PerHandleData->Socket, &(PerIoData->DataBuf), 1,
        &BytesTransferred, &Flags, &(PerIoData->Overlapped), NULL);
    }
}
```

3.4 Các phương pháp vào ra

- 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 4. MFC Socket

Chương 4. MFC Socket

4.1. Giới thiệu

4.2. CSocket

4.3. CAsyncSocket

Chương 4.1 Giới thiệu

- MFC: Microsoft Foundation Classes
- Bộ thư viện hướng đối tượng C++ lập trình ứng dụng trên Window.
- Cung cấp hai lớp hỗ trợ lập trình mạng
 - CAsyncSocket: Đóng gói lại thư viện WinSock dưới dạng hướng đối tượng. Hoạt động ở chế độ bất đồng bộ.
 - CSocket: Kế thừa từ CAsyncSocket và cung cấp giao diện ở mức cao hơn nữa. Hoạt động ở chế độ đồng bộ.
- Tập tiêu đề: **afxsock.h**

Chương 4.2 CSocket

- Khởi tạo thư viện: tự động bởi framework qua hàm `AfxSocketInit`

```
WSADATA wsaData;  
AfxSocketInit(&wsaData);
```

- Khởi tạo đối tượng CSocket: Phương thức `Create`

```
BOOL Create(  
    UINT nSocketPort = 0,           // Cổng, mặc định là 0  
    int nSocketType = SOCK_STREAM,  // Kiểu socket  
    LPCTSTR lpszSocketAddress = NULL) // Địa chỉ giao diện mạng,
```

Giá trị trả về:

- Khác NULL nếu thành công
- NULL nếu thất bại. Mã lỗi có thể truy nhập qua hàm `GetLastError()`

Ví dụ:

```
CSocket Server, Client;  
Server.Create(8888);  
Client.Create();
```

Chương 4.2 CSocket

- Kết nối đến máy khác: Phương thức Connect

```
BOOL Connect(  
    LPCTSTR lpszHostAddress,    // Địa chỉ/tên miền máy đích  
    UINT nHostPort              // Cổng  
);  
  
BOOL Connect(  
    const SOCKADDR* lpSockAddr, // Địa chỉ máy đích dưới dạng SOCKADDR  
    int nSockAddrLen            // Chiều dài cấu trúc địa chỉ  
);
```

Giá trị trả về:

- Khác NULL nếu thành công
- NULL nếu thất bại. Mã lỗi có thể truy nhập qua hàm GetLastError()

Ví dụ:

```
CSocket s;  
s.Create();  
s.Connect(L"www.google.com.vn", 80);
```

Chương 4.2 CSocket

- Đợi kết nối từ máy khác: Phương thức Listen

```
BOOL Listen( int nConnectionBacklog = 5 );
```

Giá trị trả về:

- Khác NULL nếu thành công
- NULL nếu thất bại. Mã lỗi có thể truy nhập qua hàm GetLastError()

- Đóng kết nối: Phương thức Close

```
virtual void Close();
```

Chương 4.2 CSocket

- Chấp nhận kết nối từ máy khác: Phương thức Accept

```
virtual BOOL Accept(  
    CSocket& rConnectedSocket, // Socket tương ứng với kết nối mới  
    SOCKADDR* lpSockAddr = NULL, // Địa chỉ socket mới dưới dạng SOCKADDR  
    int* lpSockAddrLen = NULL // Chiều dài địa chỉ  
);
```

Giá trị trả về:

- Khác NULL nếu thành công
- NULL nếu thất bại. Mã lỗi có thể truy nhập qua hàm GetLastError()

Ví dụ:

```
CSocket Server, Client;  
// Khởi tạo socket Server  
...  
// Chấp nhận kết nối  
Server.Accept(Client);  
// Gửi nhận dữ liệu trên Client  
...
```

Chương 4.2 CSocket

- Gửi dữ liệu đến máy khác: Phương thức Send

```
virtual int Send(  
    const void* lpBuf, // Bộ đệm chứa dữ liệu cần gửi  
    int nBufLen,       // Số byte cần gửi  
    int nFlags = 0      // Cờ, chỉ có thể là MSG_OOB nếu có  
);
```

Giá trị trả về:

- Số byte gửi được nếu thành công
- SOCKET_ERROR nếu thất bại

Ví dụ:

```
char buff[] = "Hello MFC Socket";  
Client.Send(buff, strlen(buff));
```


Chương 4.2 CSocket

- Nhận dữ liệu từ máy khác: Phương thức Receive

```
virtual int Receive(  
    void* lpBuf,           // Bộ đệm sẽ nhận dữ liệu  
    int nBufLen,           // Kích thước bộ đệm  
    int nFlags = 0         // Cờ, có thể là MSG_PEEK hoặc MSG_OOB  
);
```

Giá trị trả về:

- Số byte nhận được nếu thành công
- NULL nếu kết nối bị đóng
- SOCKET_ERROR nếu thất bại

Ví dụ:

```
...  
char buff[1024];  
int nBytesReceived = connectedSocket.Receive(buff, 1024);  
...
```

Chương 4.2 CSocket

- Ví dụ: Xây dựng Client bằng CSocket

```
...
CSocket s;
unsigned char buff[1024];
char * request = "GET / HTTP/1.1\nHost:www.google.com\n\n";
int len = 0;
s.Create();
s.Connect(L"www.google.com", 80);
s.Send(request, strlen(request));
len = s.Receive(buff, 1024);
buff[len] = 0;
printf("%s", buff);
...
```

Chương 4.2 CSocket

- Ví dụ: Xây dựng Server bằng CSocket

```
...
CSocket listen, connect;
char * buff = "Hello Network Programming";
listen.Create(80);
listen.Listen();
listen.Accept(connect);
connect.Send(buff, strlen(buff));
connect.Close();
...
```

Chương 4.3 CAsyncSocket

- Đóng gói hoạt động của socket bất đồng bộ
- Nguyên mẫu các hàm vào ra tương tự CSocket nhưng trở về ngay lập tức từ lời gọi.
- Ứng dụng không sử dụng trực tiếp lớp này mà kế thừa và chồng lên các phương thức ảo của lớp để xử lý các sự kiện.
- Các phương thức hay được chồng
 - **OnAccept**: Phương thức này sẽ được gọi mỗi khi có yêu cầu kết nối.
 - **OnClose**: Phương thức này sẽ được gọi mỗi khi socket đầu kia bị đóng.
 - **OnSend**: Phương thức này được gọi khi socket có thể gửi dữ liệu.
 - **OnReceive**: Phương thức này được gọi khi socket nhận được dữ liệu và chờ ứng dụng xử lý
 - **OnConnect**: Phương thức này được gọi khi yêu cầu kết nối được chấp nhận và socket đã sẵn sàng để gửi nhận dữ liệu.

Chương 4.3 CAsyncSocket

- Khởi tạo đối tượng: Phương thức Create

```
BOOL Create(  
    UINT nSocketPort = 0,                // Cổng  
    int nSocketType = SOCK_STREAM,       // Kiểu socket  
    long lEvent =  
    FD_READ|FD_WRITE|FD_OOB|FD_ACCEPT|FD_CONNECT|FD_CLOSE,  
                                // Mặt nạ sự kiện  
    LPCTSTR lpszSocketAddress = NULL    // Địa chỉ socket  
);
```

Giá trị trả về :

- Khác NULL nếu thành công
- NULL nếu thất bại

Sự khác biệt duy nhất với CSocket ở phương thức này là tham số lEvent chứa mặt nạ các sự kiện ứng dụng mong muốn nhận được

Chương 4.3 CAsyncSocket

- Xử lý các sự kiện: chồng lên phương thức tương ứng với sự kiện mong muốn

```
#include "afxsock.h"

class ClientSocket : public CAsyncSocket
{
public:
    ClientSocket();
    ~ClientSocket();

    void OnReceive(int nErrorCode);
};
```

Chương 4.3 CAsyncSocket

- Xử lý các sự kiện (tiếp)

```
void ClientSocket::OnReceive(int nErrorCode)
{
    char buf[1024];
    int ret;

    ret = Receive(buf, sizeof(buf));
    if (ret == SOCKET_ERROR)
    {
        Close();
        return;
    }

    if (ret == 0)
    {
        Close();
        return;
    }

    buf[ret] = 0;
    printf("%s", buf);
}
```

Chương 4.3 CAsyncSocket

- Do các sự kiện được gọi thông qua cơ chế thông điệp của Windows nên trong chương trình chính cần phải có các hàm xử lý thông điệp

```
WSADATA wsa;  
AfxSocketInit(&wsa);  
  
ClientSocket client;  
client.Create();  
client.Connect(L"127.0.0.1", 9000);  
  
MSG msg;  
while (GetMessage(&msg, NULL, 0, 0) != 0)  
{  
    TranslateMessage(&msg);  
    DispatchMessage(&msg);  
}
```


Bài tập: HTTP Server

Viết chương trình HTTP Streaming server thực hiện thao tác sau:

- Đợi kết nối ở cổng 80
- Xử lý các request từ client gửi đến có dạng

```
GET /<TenFile> HTTP/1.1
Host:
...
\n\n
```
- Phản hồi các request như sau:
 - Nếu tên <TenFile> tồn tại trong thư mục hiện tại thì gửi trả phản hồi có dạng

```
HTTP/1.1 200 OK\n
Content-Length:<KichThuocFile>\n
Content-Type:video/mp4\n
\n\n
<NoiDungFile>
```

Bài tập: HTTP Server

- Nếu file không tồn tại thì phản hồi lại như sau

```
HTTP/1.1 404 Not Found\n
```

```
Content-Type:text/html\n
```

```
\n
```

```
\n
```

```
Không tìm thấy tệp tin
```

Bài tập: Mã hóa thông điệp

Viết chương trình gửi tin nhắn mã hóa qua mạng bằng blocking. Cách thức mã hóa như sau:

- Server chọn một số nguyên x (0-255) làm mật khẩu. và gửi cho mỗi client khi kết nối đến.
- Mã ASCII của ký tự được gửi sẽ được cộng thêm x trước khi truyền, bên nhận trừ đi x để hiển thị. Nếu giá trị cộng thêm >255 thì truyền đi phần dư của giá trị đó khi chia cho 256.

Chương 5. .NET Socket

Chương 5. .NET Socket

5.1. Giới thiệu

5.2. TCP Server

5.3. TCP Client

5.4. UDP Server/Client

Chương 5.1 Giới thiệu

- .NET Framework là bộ thư viện chạy trên đa kiến trúc của Microsoft
- Hai namespace hỗ trợ lập trình mạng: System.Net và System.Net.Sockets
- Một vài lớp chính
 - IPAddress: Lưu trữ và quản lý địa chỉ IP.
 - IPEndPoint: Lưu trữ thông tin về một địa chỉ socket, tương tự như SOCKADDR_IN. Bao gồm IPAddress và cổng.
 - DNS: Hỗ trợ các thao tác phân giải tên miền
 - Socket: Xử lý các thao tác trên socket
- Tham khảo thêm

[https://msdn.microsoft.com/en-us/library/b6xa24z5\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/b6xa24z5(v=vs.110).aspx)

Chương 5.1 Giới thiệu

- **IPAddress:** Đóng gói một địa chỉ IP
 - Khởi tạo: `IPAddress.Parse("192.168.1.1");`
 - Lấy dạng chuỗi: `IPAddress.ToString();`
 - Các địa chỉ đặc biệt: `IPAddress.Any`, `IPAddress.Broadcast`, `IPAddress.Loopback`
- **EndPoint:** Đóng gói một địa chỉ socket
 - Khởi tạo: `EndPoint(IPAddress, Int32)`
 - Lấy dạng chuỗi: `EndPoint.ToString();`
- **DNS:** thực hiện phân giải tên miền
 - Lấy địa chỉ IP:
`IPAddress[] DNS.GetHostAddress("www.google.com");`

Chương 5.2 TCP Server

- Trình tự tạo TCP Server
 1. Tạo một Socket
 2. Liên kết với một IPEndPoint cục bộ
 3. Lắng nghe kết nối
 4. Chấp nhận kết nối
 5. Gửi nhận dữ liệu theo giao thức đã thiết kế
 6. Đóng kết nối sau khi đã hoàn thành và trở lại trạng thái lắng nghe chờ kết nối mới.

Chương 5.2 TCP Server

- Ví dụ

```
// Thiết lập địa chỉ của server
IPEndPoint ie = new IPEndPoint(IPAddress.Any, 8888);
// Tạo socket server
Socket server = new Socket(AddressFamily.InterNetwork,
                           SocketType.Stream, ProtocolType.Tcp);

int ret;
// Bind và Listen
server.Bind(ie);
server.Listen(10);
Console.WriteLine("Doi ket noi tu client...");
// Chấp nhận kết nối mới
Socket client = server.Accept();
Console.WriteLine("Chap nhan ket noi tu:{0}",
                  client.RemoteEndPoint.ToString());
string s = "Hello .Net Socket";
byte[] data = Encoding.ASCII.GetBytes(s);
client.Send(data, data.Length, SocketFlags.None);
```

Chương 5.2 TCP Server

- Ví dụ (tiếp)

```
while (true) {  
    data = new byte[1024];  
    ret = client.Receive(data);  
    if (ret == 0) break;  
    Console.WriteLine("Du lieu tu client:{0}",  
        Encoding.ASCII.GetString(data,0,ret));  
}  
  
client.Shutdown(SocketShutdown.Both);  
client.Close();
```

Chương 5.3 TCP Client

- Trình tự
 1. Xác định địa chỉ của Server
 2. Tạo Socket
 3. Kết nối đến Server
 4. Gửi nhận dữ liệu theo giao thức đã thiết kế
 5. Đóng Socket

Chương 5.3 TCP Client

- Ví dụ

```
// Thiết lập địa chỉ
EndPoint iep = new EndPoint(IPAddress.Parse("127.0.0.1"), 8888);

// Tạo socket client
Socket client = new Socket(AddressFamily.InterNetwork,
    SocketType.Stream, ProtocolType.Tcp);

// Kết nối đến server
client.Connect(iep);
byte[] data = new byte[1024];
int recv = client.Receive(data); // Nhận câu chào từ server
string s = Encoding.ASCII.GetString(data, 0, recv);
Console.WriteLine("Server gui:{0}", s);
string input;
while (true) {
    input = Console.ReadLine();
    //Chuyen input thanh mang byte gui len cho server
    data = Encoding.ASCII.GetBytes(input);
    client.Send(data, data.Length, SocketFlags.None);
}
```

Chương 5.3 TCP Client

- Ví dụ (tiếp)

```
        if (input.ToUpper().Equals("QUIT")) break;
    }

    client.Disconnect(true);
    client.Close();
```

Chương 5.4 UDP Server/Client

- Trình tự UDP Server/Client
 1. Tạo một Socket
 2. Liên kết với một IPEndPoint cục bộ qua hàm Bind (UDP Server) hoặc xác định địa chỉ Server để gửi dữ liệu (UDP Client)
 3. Gửi nhận dữ liệu theo giao thức đã thiết kế bằng hàm ReceiveFrom/SendTo
 4. Đóng Socket

Bài tập: Mã hóa thông điệp

- Viết chương trình gửi tin nhắn mã hóa qua mạng bằng blocking. Cách thức mã hóa như sau:
 - Server chọn một số nguyên x (0-255) làm mật khẩu. và gửi cho mỗi client khi kết nối đến.
 - Mã ASCII của ký tự được gửi sẽ được cộng thêm x trước khi truyền, bên nhận trừ đi x để hiển thị. Nếu giá trị cộng thêm >255 thì truyền đi phần dư của giá trị đó khi chia cho 256.

Báo cáo Bài tập lớn môn học

- Bài tập đã nhận
- Thành viên và phân chia công việc
- Các tài liệu sẽ đọc (tham khảo)
- Cấu trúc (block diagram, sơ đồ khối) chương trình
- Phác thảo giao diện
- Làm thế nào để trình diễn chương trình
- Mã nguồn tham khảo