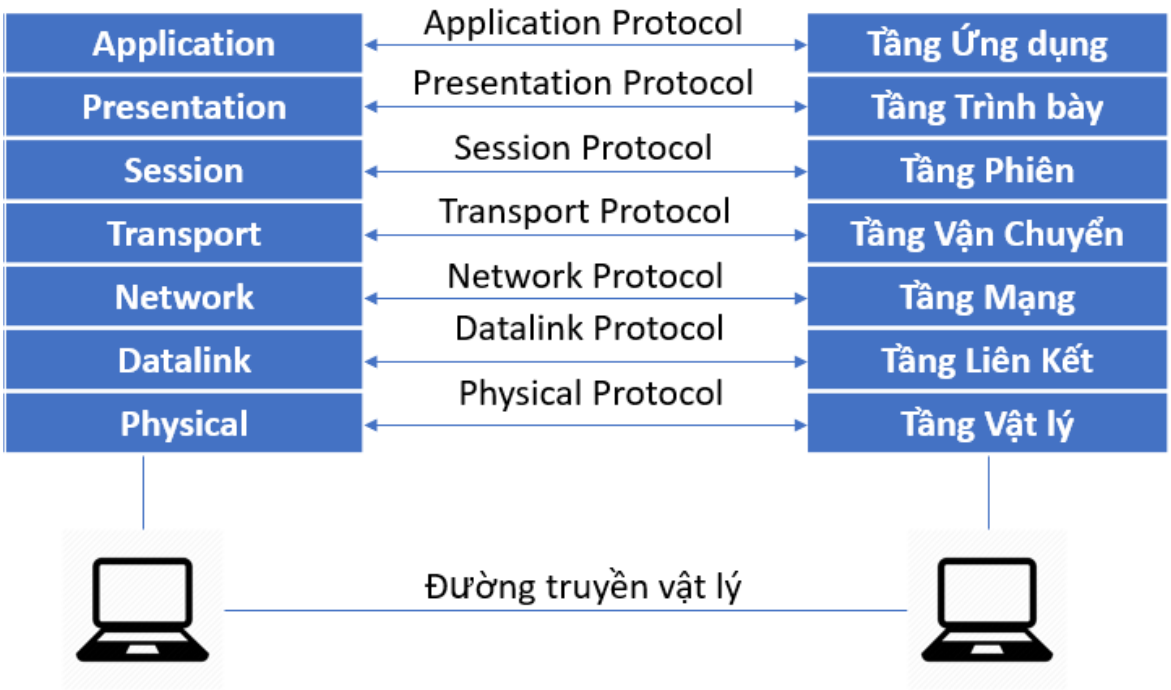
1. Mô hình OSI

[**A**nh **P**hải **S**ống **T**heo **N**gười **Đ**ịa **P**hương]

Mô hình OSI tổ chức các giao thức truyền thông thành 7 tầng, mỗi một tầng giải quyết một phần hẹp của tiến trình truyền thông, chia tiến trình truyền thông thành nhiều tầng và trong mỗi tầng có thể có nhiều giao thức khác nhau thực hiện các nhu cầu truyền thông cụ thể.

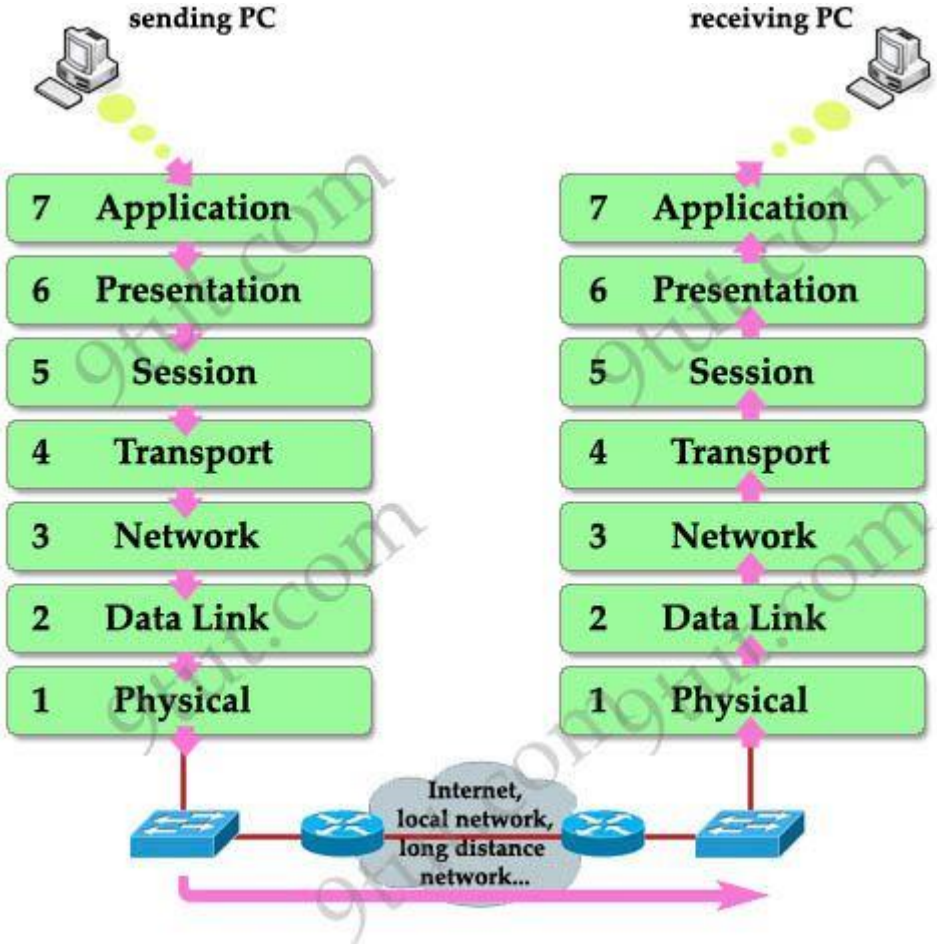


Hãy lấy một ví dụ trong thực tế cuộc sống của chúng ta để chứng minh mô hình OSI. Có lẽ bạn đã từng gửi một mail cho bạn bè của bạn, phải không? Để làm điều đó, bạn phải làm theo các bước sau:

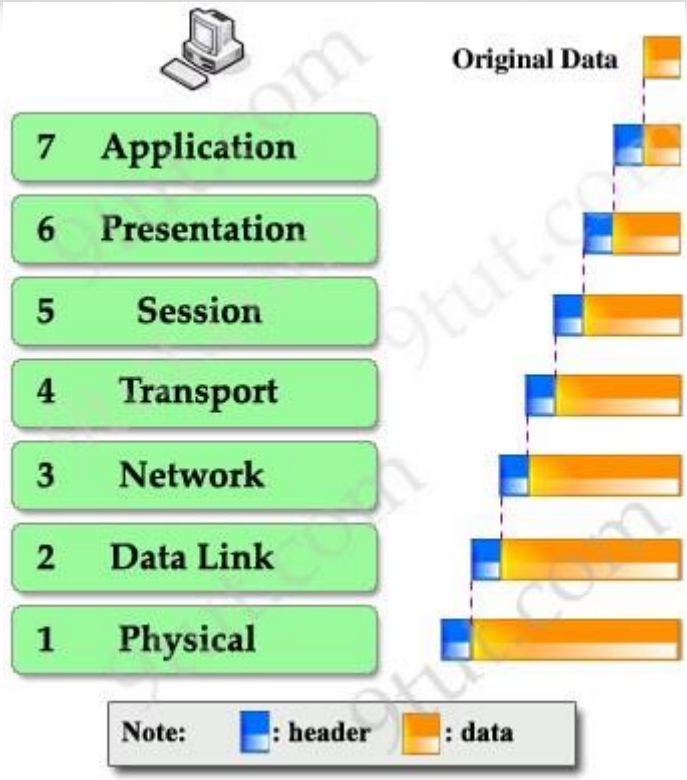
- Viết thư
- Chèn nó vào một phong bì
- Ghi thông tin về người gửi và người nhận trên phong bì mà
- Dán tem cho nó
- Đi đến bưu điện và thả nó vào một hộp thư

Từ ví dụ trên, tôi muốn ngụ ý rằng chúng ta phải đi qua một số bước theo một thứ tự cụ thể để hoàn thành một nhiệm vụ. Nó cũng được áp dụng cho hai máy tính giao tiếp với nhau. Chúng phải sử dụng một mô hình được xác định trước, cụ thể là mô hình OSI, để hoàn thành từng bước.

Khi một thiết bị muốn gửi thông tin cho thiết bị khác, dữ liệu của nó phải đi từ lớp trên xuống lớp dưới cùng (encapsulate). Nhưng khi một thiết bị nhận thông tin này, nó phải đi từ dưới lên trên để mở gói (decapsulate).



Lưu ý: Các lớp trong mô hình OSI thường được gọi bằng số thay vì tên (ví dụ, chúng ta thường gọi "lớp 3" thay vì "lớp mạng"), do đó tốt nhất bạn nên nhớ cả thứ tự của từng lớp trong mô hình OSI.



1.1. Tầng Ứng dụng

Application là lớp ở tầng trên cùng – hầu hết người dùng đều thấy và sử dụng nó. Trong mô hình OSI, đây là tầng "gần gũi với người dùng nhất". *Các ứng dụng hoạt động ở tầng thứ 7 là những ứng dụng mà người dùng tương tác trực tiếp với nó*. Nhưng chú ý rằng các chương trình bạn đang sử dụng (như một trình duyệt web – IE, Firefox hay Opera…) không thuộc về tầng Application.

Ví dụ các ứng dụng ở tầng 7: Telnet, FTP, client email (SMTP), HyperText Transfer Protocol (HTTP) là những ví dụ của lớp Application.

1.2. Tầng Trình bày

Đây là khu vực độc lập với lớp ứng dụng – nói chung, nó làm nhiệm vụ dịch chuyển định dạng tầng Application sang định dạng mạng, hoặc từ định dạng mạng sang định dạng tầng Application. Hay nói một cách khác, *lớp này thể hiện dữ liệu cho ứng dụng hoặc mạng*.

Ví dụ điển hình của tầng Trình bày là mã hóa và giải mã giữ liệu đề truyền tin an toàn như giao thức: SSL (thường bạn hay thấy địa chỉ website có dạng https://). Hay bạn muốn gửi một email và tầng trình bày sẽ định dạng dữ liệu của bạn sang định dạng

email. Hoặc bạn muốn gửi ảnh cho bạn bè của bạn, tầng Presentation sẽ định dạng dữ liệu của bạn vào các định dạng GIF, JPG hoặc PNG….

1.3. Tầng Phiên

Tầng phiên cho phép người sử dụng trên các máy khác nhau thiết lập, duy trì và đồng bộ phiên truyền thông giữa họ với nhau. Nói cách khác tầng phiên thiết lập "các giao dịch" giữa các thực thể đầu cuối. Các chức năng ở tầng này liên quan đến việc cài đặt, điều phối (ví dụ: hệ thống nên chờ phản hồi trong bao lâu) và chấm dứt kết nối giữa các ứng dụng tại cuối mỗi session.

1.4. Tầng Vận chuyển

Tầng transport liên quan đến việc phối hợp vận chuyển dữ liệu giữa các hệ thống đầu cuối. Nó có trách nhiệm đảm bảo sẽ có bao dữ liệu để gửi, tốc độ như thế nào và nó sẽ đi tới đâu.

Ví dụ dễ hiểu nhất về tầng Transport chính là giao thức TCP (Transmission Control Protocol) – được xây dựng dựa trên giao thức IP (Internet Protocol), thường được gọi là TCP/IP và UDP (User Datagram Protocol) hoạt động ở tầng 4, trong khi địa chỉ IP hoạt động ở tầng 3 (tầng Network).

1.5. Tầng Mạng

Đây là tầng network – *chức năng ở tầng này chủ yếu là định tuyến dữ liệu* – là nơi hầu hết các chuyên gia về mạng làm việc và quan tâm đến. Theo định nghĩa cơ bản nhất, tầng này chịu trách niệm cho việc chuyển tiếp các gói tin, bao gồm định tuyến thông qua rất nhiều các router khác nhau trên không gian mạng.

Ví dụ, mình ở Hồ Chí Minh và muốn truy cập đến máy chủ Facebook đặt ở Mỹ, nhưng có đến hàng triệu con đường khác nhau từ máy tính của mình tới máy chủ bên kia trái đất. Thiết bị router tại tầng này sẽ làm nhiệm vụ định tuyến cực kỳ hiệu quả giúp mình.

1.6. Tầng Liên kết/ Tầng Liên kết Dữ liệu

Các lớp liên kết dữ liệu định dạng các thông điệp vào một khung dữ liệu(Frame) và thêm vào đó một header chứa các địa chỉ phần cứng nơi nhận và địa chỉ nguồn của nó. Tiêu đề này *chịu trách nhiệm cho việc tìm kiếm các thiết bị đích tiếp theo trên một mạng nội bộ*. Chú ý rằng tầng 3 là chịu trách nhiệm cho việc tìm kiếm con đường đến đích

cuối cùng (mạng) nhưng nó không quan tâm về việc ai sẽ là người nhận tiếp theo. Vì vậy tầng 2 giúp cho dữ liệu truyền được điểm đến tiếp theo.

Tại tầng này, có thể được chia thành 2 tầng con bao gồm: lớp Media Access Control (MAC) và lớp LLC (Logical Link Control). Trong thế giới network, hầu hết các thiết bị chuyển mạch (switch) đều hoạt động ở tầng 2.

Lớp con MAC mang địa chỉ vật lý của mỗi thiết bị trên mạng.Địa chỉ này là thường được gọi là địa chỉ MAC của thiết bị. Địa chỉ MAC là một địa chỉ 48-bit được ghi vào NIC trên thiết bị của nhà sản xuất.

1.7. Tầng Vật lý

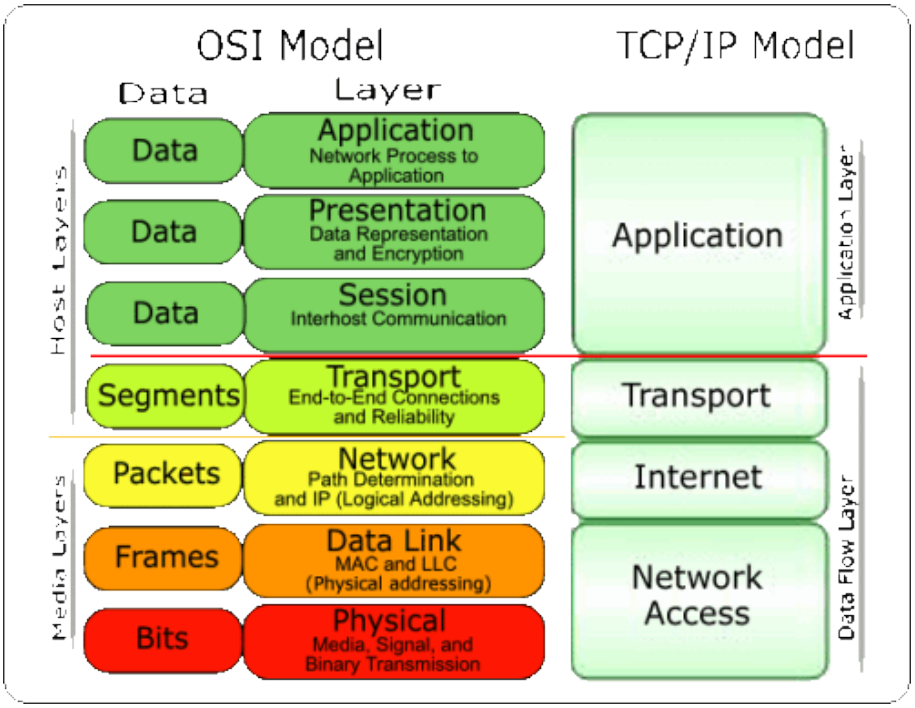
Là tầng thấp nhất trong mô hình OSI, *đây là tầng định nghĩa đặc tả về điện và vật lý cho các thiết bị của hệ thống*. Bao gồm mọi thứ từ dây cáp mạng, hệ thống mạng không dây, cũng như cách bố trí chân dây điện, hiệu điện thế và các thiết bị vật lý khác. Khi có sự cố mạng xảy ra, thường thì mọi người sẽ đi qua kiểm tra tầng physical. Ví dụ như kiểm tra các các dây cáp đã được cắm đúng chưa, kiểm tra phích cắm router, switch hay máy tính đã chuẩn chưa.

Lớp	Miêu tả	Các giao thức phổ biến	Đơn vị dữ liệu giao thức	Thiết bị hoạt động trong lớp này
Ứng dụng	+ Giao diện người dùng	HTTP, FTP, TFTP, Telnet, SNMP, DNS ...	Dữ liệu (Data,Message)	
Trình bày	+ Đại diện dữ liệu, mã hóa và giải mã	+ Video (WMV, AVI ...) + Bitmap (JPG, BMP, PNG ...) + Audio (WAV, MP3, WMA ...)	Dữ liệu (Data,Message)	
Phiên	+ Thiết lập, theo dõi và chấm dứt các phiên kết nối	+ Tên SQL, RPC, NETBIOS ...	Dữ liệu (Data,Message)	
Vận chuyển	+ Dòng điều khiển (Buffering, Windowing, Congestion Avoidance) giúp ngăn ngừa sự mất mát của các phân đoạn trên mạng và sự cần thiết phải truyền lại	+ TCP (Connection-Oriented, đáng tin cậy) + UDP (Connectionless, không đáng tin cậy)	Segment	
Mạng	+ Xác định đường dẫn + Địa chỉ logic(Nguồn/Đích)	+ IP + IPX + AppleTalk	Packet / Datagram	Router
Liên kết dữ liệu	+ Địa chỉ vật lý Bao gồm 2 lớp: + Lớp trên: Logical Link Control (LLC) + lớp dưới: Media Access Control (MAC)	+ LAN + WAN (HDLC, PPP, Frame Relay ...)	Frame	Switch, Bridge
Vật lý	Mã hóa và truyền các bit dữ liệu + Tín hiệu điện + tín hiệu vô tuyến điện	+ FDDI, Ethernet	Bit (0, 1)	Hub, Repeater ...

Lưu ý: Trong thực tế, OSI chỉ là là một mô hình lý thuyết của mạng.Các mô hình thực tế được sử dụng trong các mạng hiện đại là mô hình TCP / IP.

2. Mô hình TCP/IP

TCP/ IP (Transmission Control Protocol/ Internet Protocol - Giao thức điều khiển truyền nhận/ Giao thức liên mạng), là một bộ giao thức trao đổi thông tin được sử dụng để truyền tải và kết nối các thiết bị trong mạng Internet. TCP/IP được phát triển để mạng được tin cậy hơn cùng với khả năng phục hồi tự động.



2.1. Tầng Ứng dụng

Tầng Ứng dụng đảm nhận vai trò giao tiếp dữ liệu giữa 2 máy khác nhau thông qua các dịch vụ mạng khác nhau. Dữ liệu khi đến đây sẽ được định dạng theo kiểu Byte nối Byte, cùng với đó là các thông tin định tuyến giúp xác định đường đi đúng của một gói tin.

Ví dụ một số giao thức trao đổi dữ liệu:

- FTP (File Transfer Protocol): giao thức chạy trên nền TCP cho phép truyền các file ASCII hoặc nhị phân theo 2 chiều.
- TFTP (Trival File Transfer Protocol): giao thức truyền file chạy trên nền UDP.
- SMTP (Simple Mail Transfer Protocol): giao thức dùng để phân phối thư điện tử.
- Telnet: cho phép truy nhập từ xa để cấu hình thiết bị.
- SNMP (Simple Network Managerment Protocol): Là ứng dụng chạy trên nền UDP, cho phép quản lý và giám sát các thiết bị mạng từ xa.

- Domain Name System (DNS): Là giao thức phân giải tên miền, được sử dụng trong hỗ trợ truy nhập Internet.

2.2. Tầng Vận chuyển

Chức năng chính của tầng 3 là xử lý vấn đề giao tiếp giữa các máy chủ trong cùng một mạng hoặc khác mạng (các thiết bị đầu cuối) được kết nối với nhau thông qua bộ định tuyến.

Trong tầng này còn bao gồm 2 giao thức cốt lõi là TCP và UDP. Trong đó, TCP đảm bảo chất lượng gói tin nhưng tiêu tốn thời gian khá lâu để kiểm tra đầy đủ thông tin từ thứ tự dữ liệu cho đến việc kiểm soát vấn đề tắc nghẽn lưu lượng dữ liệu. Trái với điều đó, UDP cho thấy tốc độ truyền tải nhanh hơn nhưng lại không đảm bảo được chất lượng dữ liệu được gửi đi.

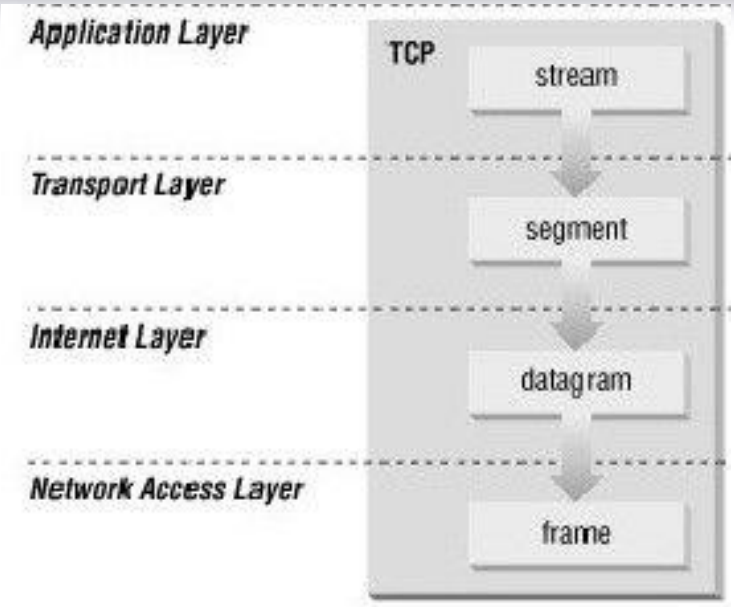
2.3. Tầng Mạng/ Tầng Internet

Nó được định nghĩa là một giao thức chịu trách nhiệm truyền tải dữ liệu một cách logic trong mạng. Các phân đoạn dữ liệu sẽ được đóng gói (Packets) với kích thước mỗi gói phù hợp với mạng chuyển mạch mà nó dùng để truyền dữ liệu. Lúc này, các gói tin được chèn thêm phần Header chứa thông tin của tầng mạng và tiếp tục được chuyển đến tầng tiếp theo.

Các giao thức chính trong tầng là IP (Internet Protocol – giao thức được sử dụng rộng rãi trong mọi hệ thống mạng trên phạm vi toàn thế giới), ICMP (Internet Control Message Protocol), IGMP (Internet Group Message Protocol).

2.4. Tầng Liên kết/ Truy cập Mạng/ Tầng Vật lý

Là sự kết hợp giữa tầng Vật lý và tầng Liên kết dữ liệu của mô hình OSI. *Chịu trách nhiệm truyền dữ liệu giữa hai thiết bị trong cùng một mạng.* Tại đây, các gói dữ liệu được đóng vào khung (gọi là Frame) và được định tuyến đi đến đích đã được chỉ định ban đầu.



Ưu điểm của mô hình TCP/IP:

- Không chịu sự kiểm soát của bất kỳ tổ chức nào => chúng ta có thể tự do trong việc sử dụng.
- Có khả năng tương thích cao với tất cả các hệ điều hành, phần cứng máy tính và mạng => hoạt động hiệu quả với nhiều hệ thống khác nhau.
- Có khả năng mở rộng cao, có thể định tuyến => có thể xác định được đường dẫn hiệu quả nhất.

3. TCP và UDP

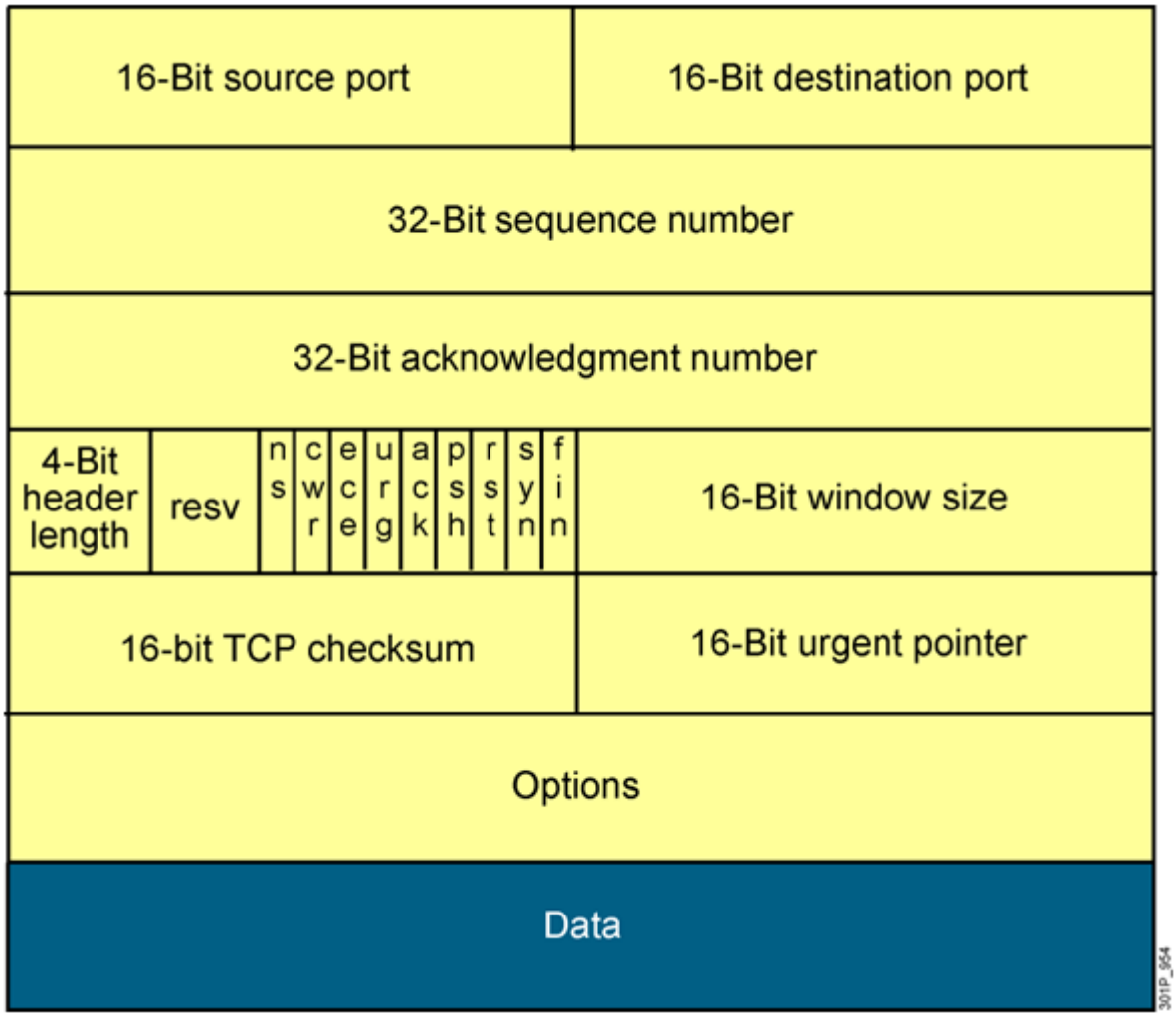
TCP và UDP đều là các giao thức được sử dụng để gửi các bit dữ liệu - được gọi là các gói tin - qua Internet. Cả hai giao thức đều được xây dựng trên giao thức IP. Nói cách khác, dù bạn gửi gói tin qua TCP hay UDP, gói này sẽ được gửi đến một địa chỉ IP. Những gói tin này được xử lý tương tự bởi vì chúng được chuyển tiếp từ máy tính của bạn đến router trung gian và đến điểm đích.

TCP và UDP không phải là giao thức duy nhất hoạt động trên IP, tuy nhiên, chúng được sử dụng rộng rãi nhất.

3.1. TCP

Giao thức TCP (Transmission Control Protocol) là giao thức hướng kết nối, nó cho phép tạo ra kết nối điểm tới điểm giữa hai thiết bị mạng, thiết lập một đường nhất quán để truyền tải dữ liệu. TCP đảm bảo dữ liệu sẽ được chuyển tới thiết bị đích, nếu dữ liệu không tới được thiết bị đích thì thiết bị gởi sẽ nhận được thông báo lỗi.

Header TCP:



- 32-bit sequence number: dùng để đánh số thứ tự gói tin (từ số sequence nó sẽ tính ra được số byte đã được truyền).
- 32-bit acknowledgement number: dùng để báo nó đã nhận được gói tin nào và nó mong nhận được byte mang số thứ tự nào tiếp theo.
- 4-bit header length: cho biết toàn bộ header dài bao nhiêu Word (1 Word = 4 byte).
- Phần kí tự (trước 16-bit Window Size): là các bit dùng để điều khiển cờ (flag) ACK, cờ Sequence v.v.
- 16-bit urgent pointer: được sử dụng trong trường hợp cần ưu tiên dữ liệu (kết hợp với bit điều khiển u r g ở trên).

IANA định nghĩa một danh sách các port TCP tiêu chuẩn được gán cho các ứng dụng đặc biệt:

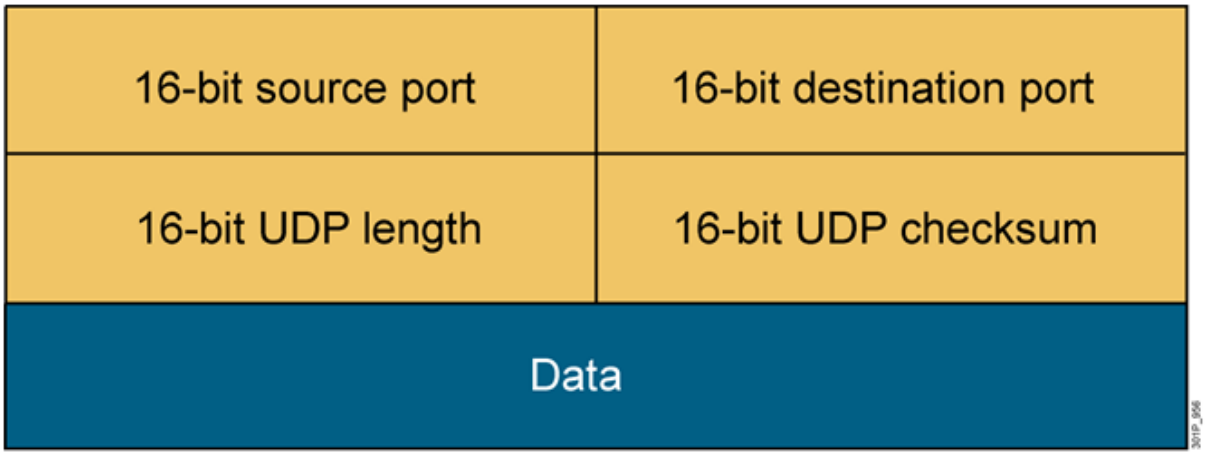
7 Echo, 13 Daytime, 17 Quote of the day, 20 FTP (data channel), 21 FTP (control channel), 22 SSH, 23 Telnet, 25 SMTP, 37 Time, 80 HTTP, 110 POP3, 119 NNTP, 123 Network Time Protocol (NTP), 137 NETBIOS name service, 138 NETBIOS datagram service, 143 Internet Message Access Protocol (IMAP), 389 Lightweight Directory Access Protocol (LDAP), 443 Secure HTTP (HTTPS), 993 Secure IMAP, 995 Secure POP3.

Các port từ 0 -> 1023 được gán cho các ứng dụng thông dụng do đó với các ứng dụng mà các lập trình viên tạo ra thì các port được gán phải từ 1024 -> 65535.

3.2. UDP

User Datagram Protocol (UDP) là một giao thức phổ biến khác được dung trong việc truyền tải dữ liệu của các gói IP. Không giống như TCP, UDP là giao thức phi nối kết. Mỗi phiên làm việc UDP không gì khác hơn là truyền tải một gói tin theo một hướng. Hình sau sẽ mô tả cấu trúc của một gói tin UDP.

Tiêu đề UDP:



- Tầng Transport dùng 1 cặp source port và destination port để định danh 1 session đang truy nhập vào đường truyền của kết nối UDP. Ta có thể coi port là địa chỉ tầng Transport (giao thức DNS chạy UDP port 53, TFTP port 69)
- 16-bit UDP Length: cho biết toàn bộ gói tin UDP dài tổng cộng bao nhiêu byte. Ta thấy 16-bit thì sẽ có tổng cộng 2^16 byte = 65536 giá trị (từ 0 -> 65535 byte).
- 16-bit UDP checksum: sử dụng thuật toán mã vòng CRC để kiểm lỗi. Và chỉ kiểm tra một cách hạn chế.

Cũng giống như TCP, UDP theo dõi các kết nối bằng cách sử dụng các port từ 1024 -> 65536, các port UDP từ 0 -> 1023 là các port dành riêng cho các ứng dụng phổ biến, một số dùng phổ biến như:

53: Domain Name System, 69: Trivial File Transfer Protocol, 111: Remote Procedure Call, 137: NetBIOS name service, 138: NetBIOS, 161: Simple Network Management Protocol.

3.3. So sánh sự khác nhau giữa TCP và UDP

TCP	UDP
Giao thức TCP có độ tin cậy cao, các gói tin được gửi bằng TCP sẽ được theo dõi do vậy dữ liệu sẽ không bị mất hoặc hỏng trong quá trình vận chuyển. Đó là lý do tại sao file tải xuống không bị hỏng ngay cả khi mạng có vấn đề. Giao thức TCP đạt được điều này theo hai cách. Đầu tiên, nó yêu cầu các gói tin bằng cách đánh số chúng. Thứ hai, nó kiểm tra lỗi bằng cách yêu cầu bên nhận gửi phản hồi đã nhận được cho bên gửi. Nếu bên gửi không nhận được phản hồi đúng, nó có thể gửi lại gói tin để đảm bảo bên nhận nhận chúng một cách chính xác. Các ứng dụng sử dụng giao thức TCP: HTTP, DNS, SMTP, telnet, SNMP v.v.	*Không có độ tin cậy cao*. Giao thức UDP hoạt động tương tự như TCP, nhưng nó bỏ qua quá trình kiểm tra lỗi. Khi một ứng dụng sử dụng giao thức UDP, các gói tin được gửi cho bên nhận và bên gửi không phải chờ để đảm bảo bên nhận đã nhận được gói tin, do đó nó lại tiếp tục gửi gói tin tiếp theo. Nếu bên nhận bỏ lỡ một vài gói tin UDP, họ sẽ mất vì bên gửi không gửi lại chúng. Do đó thiết bị có thể giao tiếp nhanh hơn. UDP được sử dụng khi tốc độ nhanh và không cần thiết sửa lỗi. Ví dụ, UDP thường được sử dụng cho các chương trình phát sóng trực tiếp, game online, DNS (cả TCP và UDP), VoIP, stream media, nghe nhạc…
TCP là giao thức hướng kết nối (connection-oriented), có nghĩa là buộc phải thiết lập kết nối trước sau đó mới đến tiến trình truyền dữ liệu.	*UDP (User Datagram Protocol) là loại giao thức phi kết nối*, không trạng thái, không cần thiết lập các kết nối trước khi gửi gói tin.

Cơ chế điều khiển luồng trong TCP (Flow Control). Giả sử: Sender gửi quá nhiều dữ liệu cho Receiver, thì Receive sẽ chuyển vào bộ đệm để chờ xử lý, đến lúc bộ đệm đầy thì B gởi tín hiệu cho A để không truyền nữa cho đến khi B xử lý hết thì sẽ gởi lại gói tin cho A để tiếp tục nhận dữ liệu.	*Không có Flow Control*

4. Bắt tay ba bước (Three-way Handshake)

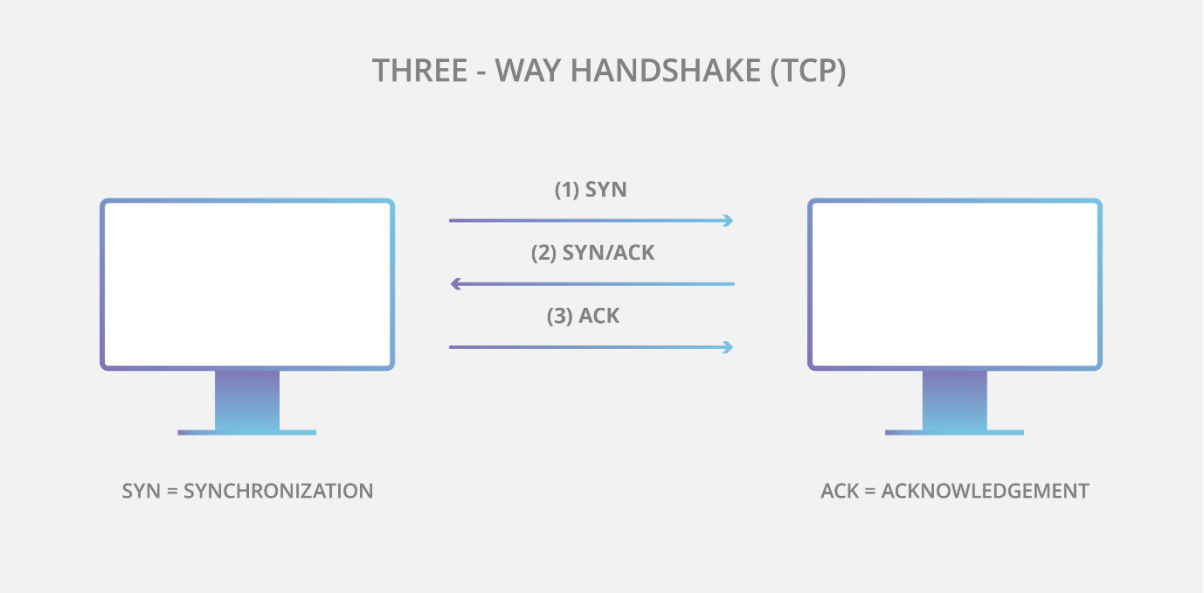
Quá trình làm thành lập một phiên làm việc TCP được thực hiện nhờ vào việc sử dụng các cờ (Flag):

- 6-bit dành riêng để sử dụng trong tương lai, giá trị luôn luôn là zero
- 1-bit URG flag: Đánh dấu gói tin là dữ liệu khẩn cấp
- 1-bit ACK flag: Hồi báo nhận một gói tin
- 1-bit PUSH flag: Cho biết dữ liệu được đẩy vào ứng dụng ngay lập tức
- 1-bit RESET flag: Thiết lập lại tình trạng khởi đầu kết nối TCP
- 1-bit SYN flag: Bắt đầu một phiên làm việc
- 1-bit FIN flag: Kết thúc một phiên làm việc

TCP sử dụng các tình trạng kết nối để quyết định tình trạng kết nối giữa các thiết bị. Một giao thức bắt tay đặc biệt được dùng để thành lập những kết nối này và theo dõi tình trạng kết nối trong suốt phiên làm việc. Một phiên làm việc TCP gồm ba pha sau:

- Bước 1: Host A gửi cho B một gói tin có cờ SYN được bật lên, với số thứ tự được đánh số là 100.
- Bước 2: Khi Host B nhận được gói tin thì gói B tiến hành gửi lại gói tin có cờ SYN đã được bật lên, kèm theo cờ ACK để xác nhận. ACK=101 có nghĩa là B thông báo cho A đã nhận được gói tin có SEQ = 100, A hãy gửi tiếp cho tôi gói tin có SEQ=101. Khi A gửi gói tin đi theo yêu cầu thì nó sẽ được đánh số thứ tự SEQ=300.
- Bước 3: Kết nối sẽ được thiết lập ngay và A sẽ gửi gói tin theo đúng yêu cầu của B. Gói tin được đánh số SEQ = 100: để đáp ứng nhu cầu của B. ACK = 301 dùng

để thông báo là đã nhận được gói tin có SEQ = 300. Chỉ có cờ ACK được bật lên bởi gói tin bước 3 sử dụng để báo nhận cho gói tin bước 2.

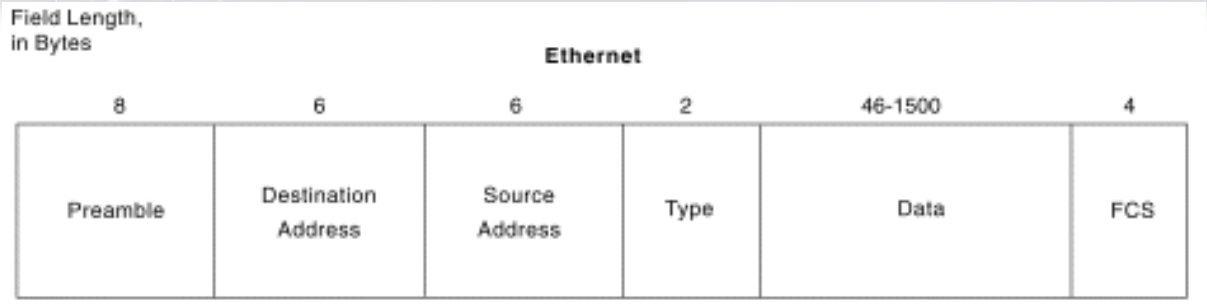


5. Giao thức Ethernet

Ethernet là một dạng công nghệ truyền thống dùng để kết nối các mạng LAN cục bộ, cho phép các thiết bị có thể giao tiếp với nhau thông qua một giao thức - một bộ quy tắc hoặc ngôn ngữ mạng chung. Là một lớp giao thức data-link trong tầng TCP/IP, Ethernet cho thấy các thiết bị mạng có thể định dạng và truyền các gói dữ liệu như thế nào, sao cho các thiết bị khác trên cùng phân khúc mạng cục bộ có thể phát hiện, nhận và xử lý các gói dữ liệu đó. Cáp Ethernet là một hệ thống dây vật lý để truyền dữ liệu qua.

So với công nghệ mạng LAN không dây, Ethernet thường ít bị gián đoạn hơn - cho dù là do nhiễu sóng vô tuyến, trở ngại vật lý hay băng thông. Ethernet cũng cung cấp mức độ bảo mật và kiểm soát mạng tốt hơn so với công nghệ không dây (các thiết bị phải được kết nối bằng cáp vật lý - người ngoài sẽ gặp khó khăn khi truy cập dữ liệu mạng hay khi cố gắng điều hướng băng thông cho các thiết bị không được cung cấp.

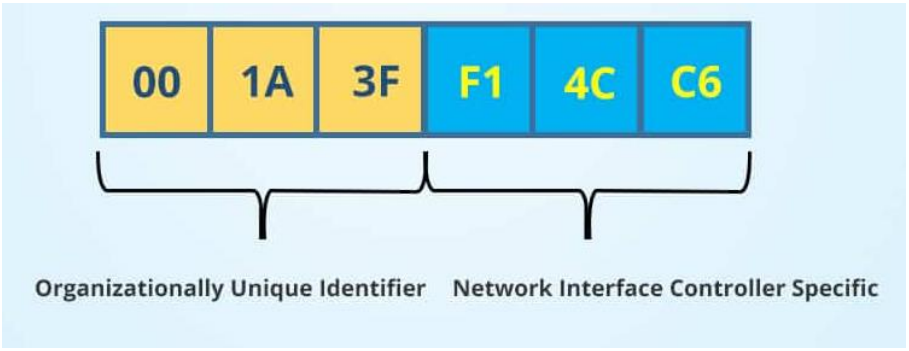
Ethernet chia dữ liệu thành nhiều khung (frame). Khung là một gói thông tin được truyền như một đơn vị duy nhất. Khung trong Ethernet có thể dài từ 64 đến 1518 byte, nhưng bản thân khung Ethernetđã sử dụng ít nhất 18 byte, nên dữ liệu một khung Ethernet có thể dài từ 46 đến 1500 byte. Mỗi khung đều có chứa thông tin điều khiển và tuân theo một cách tổ chức cơ bản.



- Preamble: 8-byte báo hiệu bắt đầu một frame.
- Destination: 6-byte thể hiện địa chỉ MAC đích.
- Source: 6-byte thể hiện địa chỉ MAC nguồn.
- Type: 2-byte chỉ rõ giao thức lớp mạng.
- Data: dữ liệu được chuyển đi.
- CRC: 4-byte kiểm tra lỗi của Frame. (Cyclic Redundancy Check)/ Frame Checksum

Địa chỉ Ethernet (địa chỉ MAC) là địa chỉ của các thiết bị, địa chỉ này được gán bởi các nhà sản xuất thiết bị mạng và nó không thay đổi được. Mỗi thiết bị trên mạng Ethernet phải có 1 địa chỉ MAC duy nhất. Địa chỉ MAC gồm 2 phần:

- 3-byte xác định nhà sản xuất
- 3-byte xác định số serial duy nhất của nhà sản xuất



Đối với địa chỉ broadcast thì tất cả các bit của địa chỉ đích được gán bằng 1 (FF-FF-FF-FF-FF-FF). Mỗi thiết bị mạng sẽ chấp nhận các gói có địa chỉ broadcast.

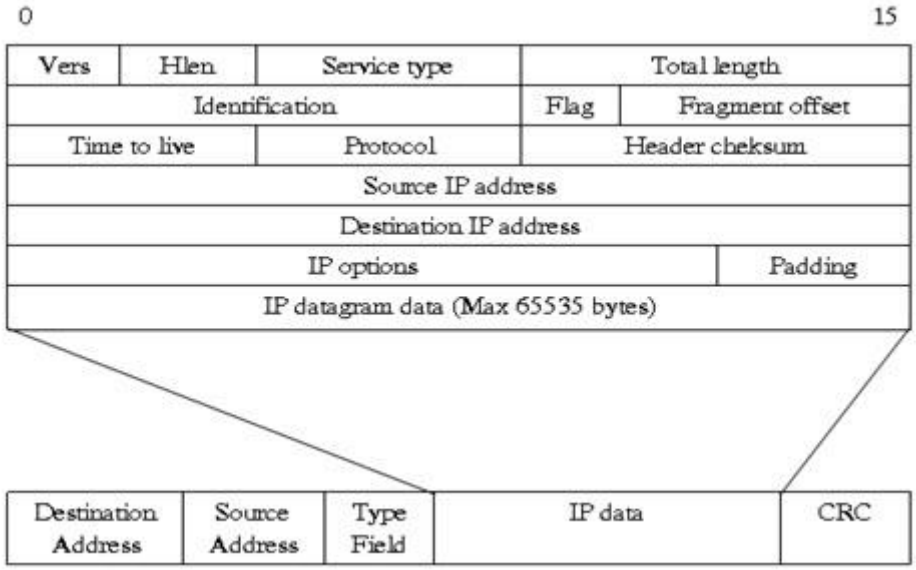
6. Giao thức IP

Internet Protocol (IP) là một giao thức liên mạng hoạt động ở tầng Network trong mô hình OSI. IP quy định cách thức định địa chỉ các máy tính và cách thức chuyển tải các gói tin qua một liên mạng. IP được đặc tả trong bảng báo cáo kỹ thuật có tên Request for Comments (RFC). IP có hai chức năng chính: cung cấp dịch vụ truyền tải các gói

tin qua một liên mạng, phân mảnh và hợp lại các gói tin. Các gói dữ liệu xuất phát từ tầng Application, đến tầng Network được thêm vào một cấu trúc IP Header. Gói dữ liệu sau khi được thêm vào cấu trúc IP Header thì được gọi là IP Diagram (Packet). Hiện nay, có hai phiên bản IP là IP Version 4 (IPv4) và IP Version 6 (IPv6), do đó có 2 cấu trúc tương ứng là IP Header Version 4 và IP header Version 6.

6.1. IPv4

IP Header Version 4 (tiêu đề gói tin IPv4) gồm 12 trường bắt buộc với tổng chiều dài là 20 byte (không tính các trường Options và Data). Cấu trúc của IP Header Version 4 như hình sau:

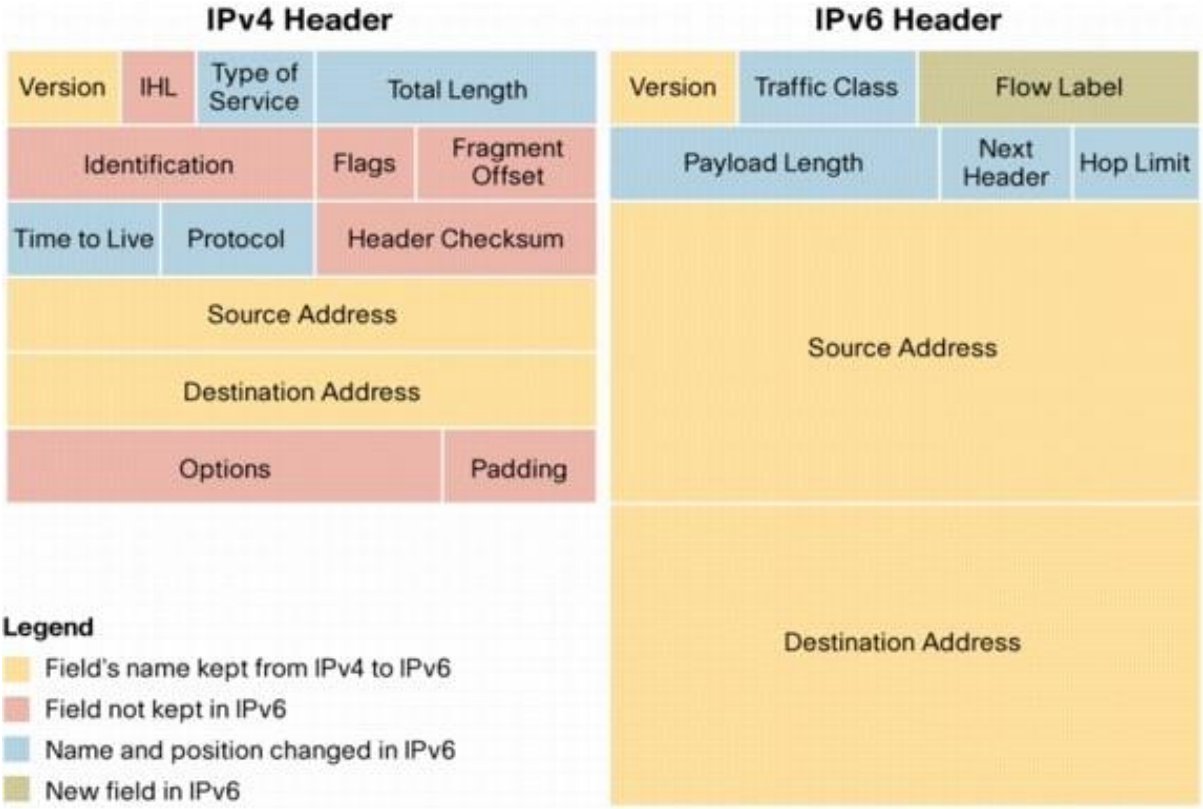


Địa chỉ Ethernet dùng để xác định các thiết bị trên mạng LAN nhưng nó không thể dùng để xác định địa chỉ của các thiết bị trên mạng ở xa. Để xác định các thiết bị trên các mạng khác nhau, địa chỉ IP được dùng. Một địa chỉ IP là một số 32-bit và địa chỉ IP được chia thành 4 lớp sau:

Lớp A	0.x.x.x–127.x.x.x
Lớp B	128.x.x.x–191.x.x.x
Lớp C	192.x.x.x–223.x.x.x
Lớp D	224.x.x.x–254.x.x.x

6.2. IPv6

Cấu trúc của IP Header Version 6 (tiêu đề gói tin IPv6) bao gồm vài trường có chức năng giống như IP Header Version 4 (nhưng tên các trường đã thay đổi) kết hợp thêm trường mới. Trường mới thể hiện được hiệu quả hoạt động của IPv6 hơn so với IPv4.

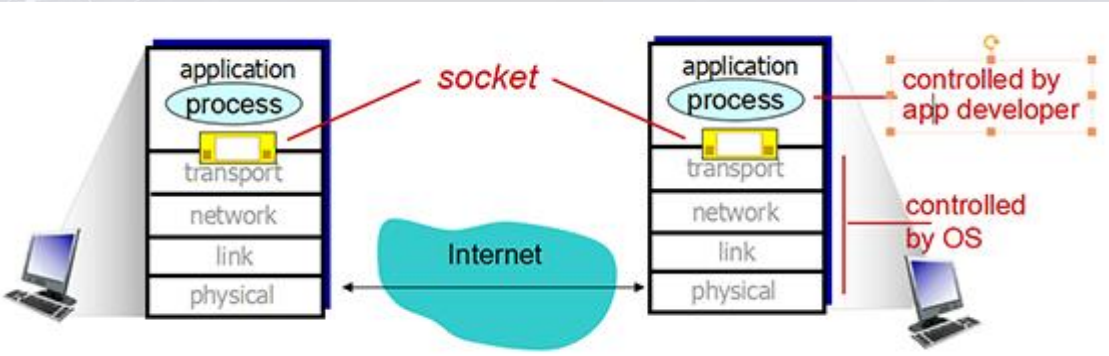


7. Socket

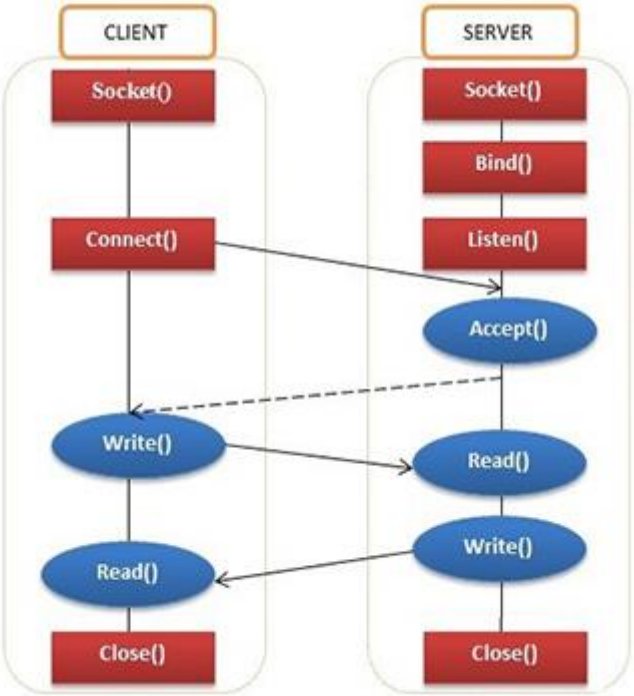
Trong lập trình mạng dùng Socket, chúng ta không trực tiếp truy cập vào các thiết bị mạng để gởi và nhận dữ liệu. Thay vì vậy, một file mô tả trung gian được tạo ra để điều khiển việc lập trình. Các file mô tả dùng để tham chiếu đến các kết nối mạng được gọi là các Socket. Socket định nghĩa những đặc trưng sau:

- Một kết nối mạng hay một đường ống dẫn để truyền tải dữ liệu
- Một kiểu truyền thông như stream (TCP), datagram (UDP), raw (hỗ trợ trừu tượng hóa socket, dành cho người dùng nâng cao muốn xây dựng một giao thức riêng), WebSocket, Sequenced packet
- Một giao thức như TCP hay UDP

Socket phải được gắn vào một địa chỉ mạng và một port trên hệ thống cục bộ hay ở xa.



7.1. Stream Socket



Stream Socket hay còn gọi là socket hướng kết nối, là socket hoạt động thông qua giao thức TCP (Transmission Control Protocol). Stream Socket chỉ hoạt động khi server và client đã kết nối với nhau.

- Các bước thiết lập một socket phía client gồm:

- Tạo một socket bằng hàm socket()
- Kết nối socket đến địa chỉ của server bằng hàm connect()
- Gửi và nhận dữ liệu: Có một số cách khác nhau, đơn giản nhất là sử dụng các hàm read() và write()
- Đóng kết nối bằng hàm close()

- Các bước thiết lập một socket phía server gồm:

- Tạo một socket bằng hàm socket()

- Gắn (bind) socket đến địa chỉ của server sử dụng hàm bind().
- Đối với server trên internet địa chỉ bao gồm địa chỉ ip của máy host + số hiệu cổng dịch vụ (port number)
- Lắng nghe (listen) các kết nối đến từ clients sử dụng hàm listen()
- Chấp nhận các kết nối sử dụng hàm accept(). Hàm này sẽ dừng (block) cho đến khi nhận được một client kết nối đến.
- Gửi và nhận dữ liệu với client (hàm read(), write())
- Đóng kết nối bằng hàm close()

- Ưu điểm của Stream Socket là gì?

- Dữ liệu truyền đi được đảm bảo truyền đến đúng nơi nhận, đúng thứ tự với thời gian nhanh chóng
- Mỗi thông điệp gửi đi đều có xác nhận trả về để thông báo cho người dùng thông tin về quá trình truyền tải.

- Nhược điểm của Stream Socket là gì?

- Giữa máy chủ và máy nhận chỉ có 1 IP, nên khi kết nối, 1 máy phải chờ máy còn lại chấp nhận kết nối.

Ví dụ TCP Server:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpServerDonGian
{
    public static void Main()
    {
        //Số byte thực sự nhận được dùng hàm Receive()
        int byteReceive;
        //buffer để nhận và gởi dữ liệu
        byte[] buff = new byte[1024];
        //EndPoint cục bộ
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        //Kết nối server với 1 EndPoint
```

```
            server.Bind(ipep);
            //Server lắng nghe tối đa 10 kết nối
            server.Listen(10);
            Console.WriteLine("Dang cho Client ket noi den...");
            //Hàm Accept() sẽ block server lại cho đến khi có Client kết nối đến
            Socket client = server.Accept();
            //Client EndPoint
            IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
            Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
            clientep.Address, clientep.Port);
            string welcome = "Hello Client";
            //Chuyển chuỗi thành mảng các byte
            buff = Encoding.ASCII.GetBytes(welcome);
            //Gởi câu chào cho Client
            client.Send(buff, buff.Length, SocketFlags.None);
            while (true)
            {
                //Reset lại buffer
                buff = new byte[1024];
                //Lấy số byte thực sự nhận được
                byteReceive = client.Receive(buff);
                //Nếu Client ngắt kết nối thì thoát khỏi vòng lặp
                if (byteReceive == 0)
                    break;
                Console.WriteLine(Encoding.ASCII.GetString(buff, 0, byteReceive));
                //Sau khi nhận dữ liệu xong, gởi lại cho Client
                client.Send(buff, byteReceive, SocketFlags.None);
            }
            Console.WriteLine("Da dong ket noi voi Client: {0}", clientep.Address);
            //Đóng kết nối
            client.Close();
            server.Close();
        }
}
```

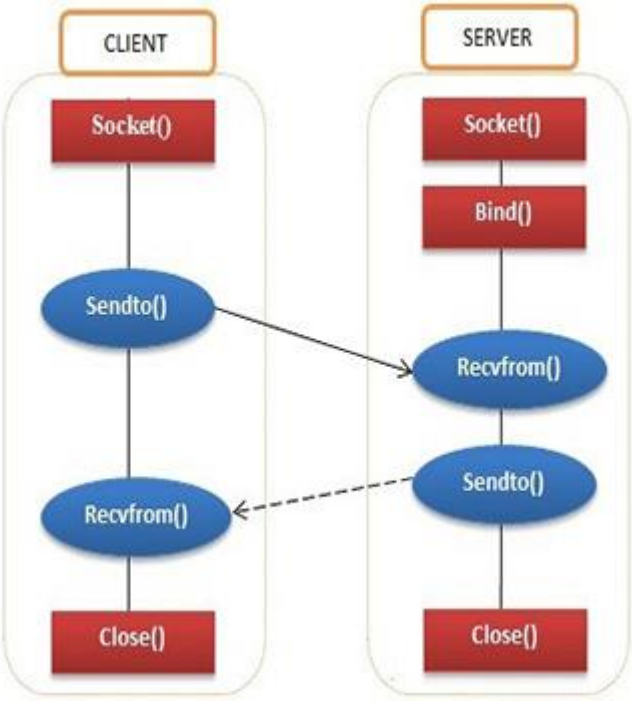
TCP Client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleTcpClient
{
```

```
    public static void Main()
    {
        //Buffer để gởi và nhận dữ liệu
        byte[] buff = new byte[1024];
        //Chuỗi nhập vào và chuỗi nhận được
        string input, stringData;
        //IPEndPoint ở server
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        //Server Socket
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        //Hàm Connect() sẽ bị block lại và chờ khi kết nối được với server thì mới hết
block
        try
        {
            server.Connect(ipep);
        }
        //Quá trình kết nối có thể xảy ra lỗi nên phải dùng try, catch
        catch (SocketException e)
        {
            Console.WriteLine("Không thể kết nối đến Server");
            Console.WriteLine(e.ToString());
            return;
        }
        //Số byte thực sự nhận được
        int byteReceive = server.Receive(buff);
        //Chuỗi nhận được
        stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
        Console.WriteLine(stringData);
        while (true)
        {
            //Nhập dữ liệu từ bàn phím
            input = Console.ReadLine();
            //Nếu nhập exit thì thoát và đóng Socket
            if (input == "exit")
                break;
            //Gởi dữ liệu cho server
            server.Send(Encoding.ASCII.GetBytes(input));
            //Reset lại buffer
            buff = new byte[1024];
            //Số byte thực sự nhận được
```

```
            byteReceive = server.Receive(buff);
            //Chuỗi nhận được
            stringData = Encoding.ASCII.GetString(buff, 0, byteReceive);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dong ket noi voi server...");
        //Dừng kết nối, không cho phép nhận và gởi dữ liệu
        server.Shutdown(SocketShutdown.Both);
        //Đóng Socket
        server.Close();
    }
}
```

7.2. Datagram Socket



Datagram Socket hay còn gọi là socket phi kết nối, là socket hoạt động thông qua giao thức UDP (User Datagram Protocol). Datagram Socket có thể hoạt động kể cả khi không có sự thiết lập kết nối giữa 2 máy với nhau.

- Ưu điểm của Datagram Socket là gì?

- Quá trình kết nối và truyền tải thông tin đơn giản, không cần thực hiện nhiều thao tác.
- Thời gian truyền tải dữ liệu cực nhanh.

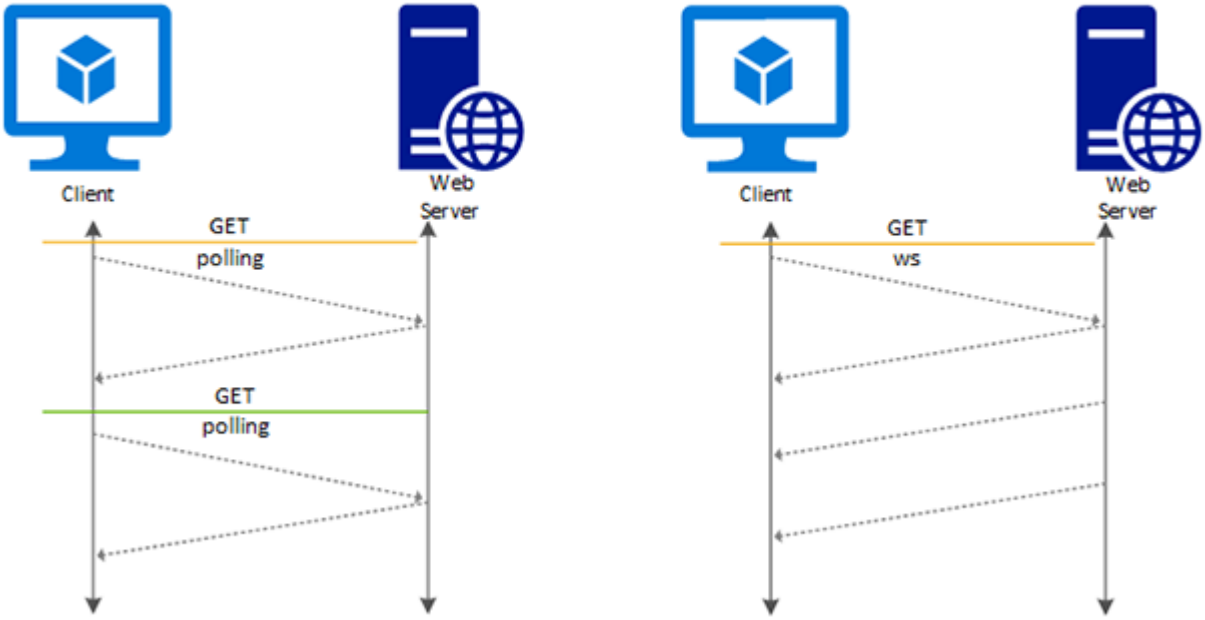
- Nhược điểm của Datagram Socket là gì?

- Quá trình truyền thông tin không đảm bảo tin cậy, thông tin có thể truyền sai thứ tự hoặc bị lặp.

Chương trình UDP Client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Hello server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref Remote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:", Remote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            server.SendTo(Encoding.ASCII.GetBytes(input), Remote);
            data = new byte[1024];
            recv = server.ReceiveFrom(data, ref Remote);
            stringData = Encoding.ASCII.GetString(data, 0, recv);
            Console.WriteLine(stringData);
        }
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

7.3. WebSocket



Websocket là công cụ hỗ trợ việc kết nối qua lại trên internet giữa client và server. Giúp diễn ra nhanh chóng và hiệu quả hơn thông qua việc sử dụng TCP socket. Không chỉ sử dụng riêng cho ứng dụng web, Websocket có thể áp dụng cho bất kì ứng dụng nào khác cần có sự trao đổi thông tin trên Internet.

- Ưu điểm của Websocket là gì?

Websocket mang lại nhiều ưu điểm trong việc kết nối giữa client và server. Cụ thể như sau:

- Tăng tốc độ truyền tải thông tin giữa 2 chiều
- Dễ phát hiện và xử lý trong trường hợp có lỗi xảy ra
- Dễ dàng sử dụng, không cần cài đặt thêm các phần mềm bổ sung khác
- Không cần sử dụng nhiều phương pháp kết nối khác nhau

- Nhược điểm của Websocket là gì?

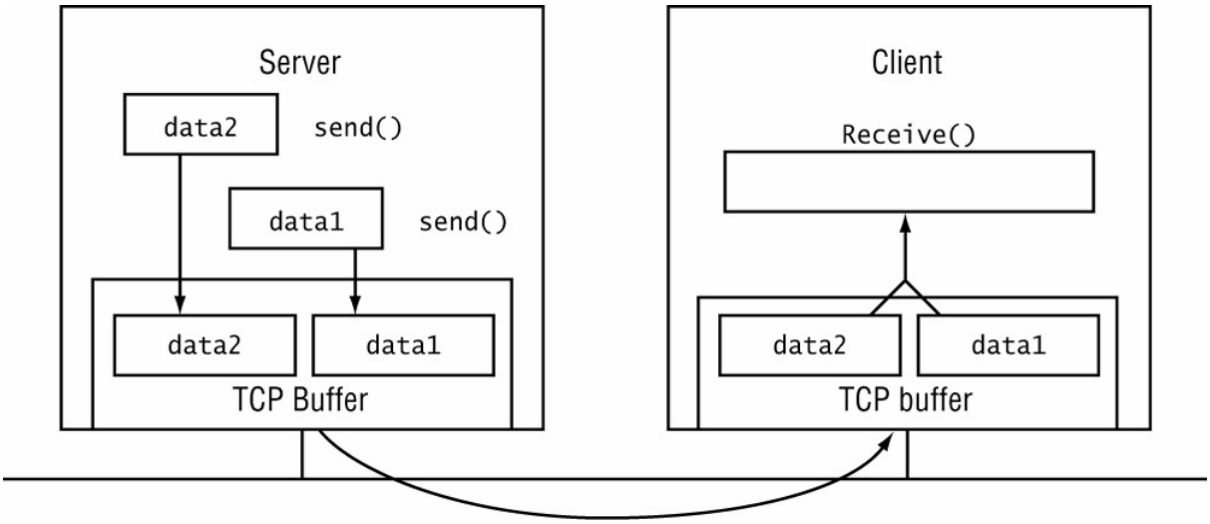
Một số nhược điểm của Websocket mà bạn cần lưu ý khi sử dụng có thể kể đến như:

- Chưa hỗ trợ trên tất cả các trình duyệt
- Với các dịch vụ có phạm vi yêu cầu, Websocket chưa hỗ trợ hoàn toàn.

8. Một số vấn đề trong lập trình

8.1. Vấn đề về bộ đệm có kích thước nhỏ

Hệ điều hành Window dùng bộ đệm TCP để gởi và nhận dữ liệu. Điều này là cần thiết để TCP có thể gởi lại dữ liệu bất cứ lúc nào cần thiết. Một khi dữ liệu đã được hồi báo nhận thành công thì nó mới được xóa khỏi bộ đệm.



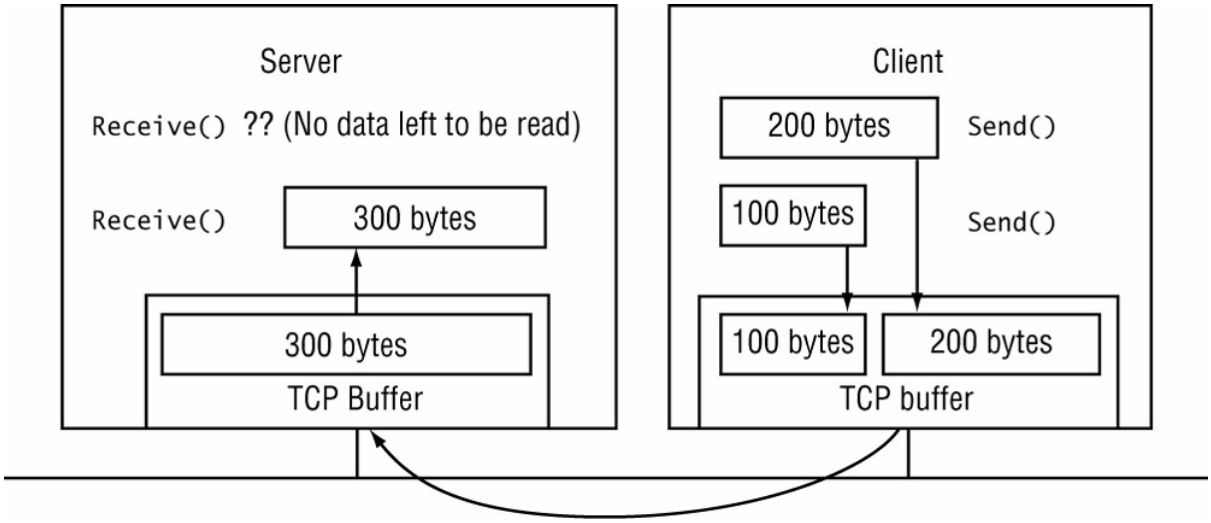
Dữ liệu đến cũng được hoạt động theo cách tương tự. Nó sẽ ở lại trong bộ đệm cho đến khi phương thức Receive() được dùng để đọc nó. Nếu phương thức Receive() không đọc toàn bộ dữ liệu ở trong bộ đệm, phần còn lại vẫn được nằm ở đó và chờ phương thức Receive() tiếp theo được đọc. Dữ liệu sẽ không bị mất nhưng chúng ta sẽ không lấy được các đoạn dữ liệu mình mong muốn.

Bởi vì bộ đệm dữ liệu không đủ lớn để lấy hết dữ liệu ở bộ đệm TCP nên phương thức Receive() chỉ có thể lấy được một lượng dữ liệu có độ lớn đúng bằng độ lớn của bộ đệm dữ liệu, phần còn lại vẫn nằm ở bộ đệm TCP và nó được lấy khi gọi lại phương thức Receive(). Do đó câu chào Client của Server phải dùng tới hai lần gọi phương thức Receive() mới lấy được hết. Trong lần gởi và nhận dữ liệu kế tiếp, đoạn dữ liệu tiếp theo được đọc từ bộ đệm TCP do đó nếu ta gởi dữ liệu với kích thước lớn hơn 10 byte thì khi nhận ta chỉ nhận được 10 byte đầu tiên.

Bởi vì vậy nên trong khi lập trình mạng chúng ta phải quan tâm đến việc đọc dữ liệu từ bộ đệm TCP một cách chính xác. Bộ đệm quá nhỏ có thể dẫn đến tình trạng thông điệp nhận sẽ không khớp với thông điệp gởi, ngược lại bộ đệm quá lớn sẽ làm cho các thông điệp bị trộn lại, khó xử lý. Việc khó nhất là làm sao phân biệt được các thông điệp được đọc từ Socket.

8.2. Vấn đề với các thông điệp TCP

Một trong những khó khăn của những nhà lập trình mạng khi sử dụng giao thức TCP để chuyển dữ liệu là giao thức này không quan tâm đến biên dữ liệu.



Như trên hình vấn đề xảy ra khi truyền dữ liệu là không đảm bảo được mỗi phương thức Send() sẽ không được đọc bởi một phương thức Receive(). Tất cả dữ liệu được đọc từ phương thức Receive() không thực sự được đọc trực tiếp từ mạng mà nó được đọc từ bộ đệm TCP. Khi các gói tin TCP được nhận từ mạng sẽ được đặt theo thứ tự trong bộ đệm TCP. Mỗi khi phương thức Receive() được gọi, nó sẽ đọc dữ liệu trong bộ đệm TCP, không quan tâm đến biên dữ liệu.

Giải quyết các vấn đề với thông điệp TCP:

Để giải quyết vấn đề với biên dữ liệu không được bảo vệ, chúng ta phải tìm hiểu một số kỹ thuật để phân biệt các thông điệp. Ba kỹ thuật thông thường dùng để phân biệt các thông điệp được gởi thông qua TCP:

- Luôn luôn sử dụng các thông điệp với kích thước cố định

Cách dễ nhất nhưng cũng là cách tốn chi phí nhất để giải quyết vấn đề với các thông điệp TCP là tạo ra các giao thức luôn luôn truyền các thông điệp với kích thước cố định. Bằng cách thiết lập tất cả các thông điệp có cùng kích thước, chương trình TCP nhận có thể biết toàn bộ thông điệp được gởi từ Client.

```
private static int SendData(Socket s, byte[] data)
{
    int total = 0;
    int size = data.Length;
    int dataleft = size;
    int sent;
    while (total < size)
    {
        sent = s.Send(data, total, dataleft, SocketFlags.None);
```

```
        total += sent;
        dataleft -= sent;
    }
    return total;
}

private static byte[] ReceiveData(Socket s, int size)
{
    int total = 0;
    int dataleft = size;
    byte[] data = new byte[size];
    int recv;
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
}
```

- Gởi kèm kích thước thông điệp cùng với mỗi thông điệp

Cách giải quyết vấn đề biên thông điệp của TCP bằng cách sử dụng các thông điệp với kích thước cố định là một giải pháp lãng phí bởi vì tất cả các thông điệp đều phải cùng kích thước. Nếu các thông điệp nào chưa đủ kích thước thì phải thêm phần đệm vào, gây lãng phí băng thông mạng.

Một giải pháp cho vấn đề cho phép các thông điệp được gởi với các kích thước khác nhau là gởi kích thước thông điệp kèm với thông điệp. Bằng cách này thiết bị nhận sẽ biết được kích thước của mỗi thông điệp.

```
private static int SendVarData(Socket s, byte[] buff)
{
    int total = 0;
    int size = buff.Length;
    int dataleft = size;
    int sent;
    byte[] datasize = new byte[4];
    datasize = BitConverter.GetBytes(size);
    sent = s.Send(datasize);
    while (total < size)
    {
        sent = s.Send(buff, total, dataleft, SocketFlags.None);
        total += sent;
        dataleft -= sent;
    }
    return total;
}

private static byte[] ReceiveVarData(Socket s)
{
```

```
    int total = 0;
    int recv;
    byte[] datasize = new byte[4];
    recv = s.Receive(datasize, 0, 4, 0);
    int size = BitConverter.ToInt32(datasize, 0);
    int dataleft = size;
    byte[] data = new byte[size];
    while (total < size)
    {
        recv = s.Receive(data, total, dataleft, 0);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit ");
            break;
        }
        total += recv;
        dataleft -= recv;
    }
    return data;
}
```

- Sử dụng các hệ thống đánh dấu để phân biệt các thông điệp

Một cách khác để gởi các thông điệp với kích thước khác nhau là sử dụng các hệ thống đánh dấu. Hệ thống này sẽ chia các thông điệp bởi các ký tự phân cách để báo hiệu kết thúc thông điệp. Khi dữ liệu được nhận từ Socket, dữ liệu được kiểm tra từng ký tự một để phát hiện các ký tự phân cách, khi các ký tự phân cách được phát hiện thì dữ liệu trước ký tự phân cách chính là một thông điệp và dữ liệu sau ký tự phân cách sẽ bắt đầu một thông điệp mới.

Phương pháp này có một số hạn chế, nếu thông điệp lớn nó sẽ làm giảm tốc độ của hệ thống vì toàn bộ các ký tự của thông điệp đều phải được kiểm tra. Cũng có trường hợp một số ký tự trong thông điệp trùng với các ký tự phân cách và thông điệp này sẽ bị tách ra thành các thông điệp con, điều này làm cho chương trình chạy bị sai lệch.

9. Sử dụng C# Stream với TCP

Điều khiển thông điệp dùng giao thức TCP thường gây ra khó khăn cho các lập trình viên nên .NET Framework cung cấp một số lớp để giảm gánh nặng lập trình. Một trong những lớp đó là NetworkStream, và hai lớp dùng để gởi và nhận text sử dụng giao thức TCP là StreamWriter và StreamReader.

TCP Client sử dụng Network Stream:

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpClient
```

```
{
    public static void Main()
    {
        string data;
        string input;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        try
        {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Khong the ket noi den server");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        data = sr.ReadLine();
        Console.WriteLine(data);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit")
                break;
            sw.WriteLine(input);
            sw.Flush();
            data = sr.ReadLine();
            Console.WriteLine(data);
        }
        Console.WriteLine("Dang dong ket noi voi server...");
        sr.Close();
        sw.Close();
        ns.Close();
        server.Shutdown(SocketShutdown.Both);
        server.Close();
    }
}
```

TCP Server sử dụng Network Stream:

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;

class StreamTcpSrvr
{
    public static void Main()
    {
        string data;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Dang cho Client ket noi toi...");
        Socket client = newsock.Accept();
```

```
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Da ket noi voi Client {0} tai port {1}",
        newclient.Address, newclient.Port);
        NetworkStream ns = new NetworkStream(client);
        StreamReader sr = new StreamReader(ns);
        StreamWriter sw = new StreamWriter(ns);
        string welcome = "Hello Client";
        sw.WriteLine(welcome);
        sw.Flush();
        while (true)
        {
            try
            {
                data = sr.ReadLine();
            }
            catch (IOException)
            {
                break;
            }
            Console.WriteLine(data);
            sw.WriteLine(data);
            sw.Flush();
        }
        Console.WriteLine("Da dong ket noi voi Client {0}", newclient.Address);
        sw.Close();
        sr.Close();
        ns.Close();
    }
}
```

10. Một số vấn đề với UDP

Một trong những tính năng quan trọng của UDP mà TCP không có được đó là khả năng xử lý thông điệp mà không cần quan tâm đến biên thông điệp. UDP bảo vệ biên thông điệp của tất cả các thông điệp được gởi. Mỗi lần gọi phương thức ReceiveFrom() nó chỉ đọc dữ liệu được gởi từ một phương thức SendTo().

UDP Server:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Dang cho client ket noi den...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)(sender);
        recv = newsock.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
        tmpRemote.ToString());
```

```
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        string welcome = "Xin chao client";
        data = Encoding.ASCII.GetBytes(welcome);
        newsock.SendTo(data, data.Length, SocketFlags.None, tmpRemote);
        for (int i = 0; i < 5; i++)
        {
            data = new byte[1024];
            recv = newsock.ReceiveFrom(data, ref tmpRemote);
            Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        }
        newsock.Close();
    }
}
```

UDP Client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TestUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 5000);
        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Xin chao Server";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint tmpRemote = (EndPoint)sender;
        data = new byte[1024];
        int recv = server.ReceiveFrom(data, ref tmpRemote);
        Console.WriteLine("Thong diep duoc nhan tu {0}:",
        tmpRemote.ToString());
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 1"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 2"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 3"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 4"), tmpRemote);
        server.SendTo(Encoding.ASCII.GetBytes("Thong diep 5"), tmpRemote);
        Console.WriteLine("Dang dong client");
        server.Close();
    }
}
```

Vì UDP không quan tâm đến việc gởi lại các gói tin nên nó không dùng bộ đệm. Tất cả dữ liệu được gởi từ Socket đều được lập tức gởi ra ngoài mạng và tất cả dữ liệu được nhận từ mạng lập tức được chuyển cho phương thức ReceiveFrom() trong lần gọi tiếp theo. Khi phương thức ReceiveFrom() được dùng trong chương trình, các lập trình viên phải đảm bảo rằng bộ đệm phải đủ lớn để chấp nhận hết dữ liệu từ UDP Socket. Nếu bộ đệm quá nhỏ, dữ liệu sẽ bị mất.

Một khó khăn khác khi lập trình với giao thức udp là khả năng bị mất gói tin bởi vì udp là một giao thức phi kết nối nên không có cách nào mà thiết bị gởi biết được gói tin gởi có thực sự đến được đích hay không. Cách đơn giản nhất để ngăn chặn việc mất các gói tin là phải có cơ chế hồi báo giống như giao thức TCP. Các gói tin được gởi thành công đến thiết bị nhận thì thiết bị nhận phải sinh ra gói tin hồi báo cho thiết bị gởi biết đã nhận thành công. Nếu gói tin hồi báo không được nhận trong một khoảng thời gian nào đó thì thiết bị nhận sẽ cho là gói tin đó đã bị mất và gởi lại gói tin đó.

11. Sử dụng TCPClient, TCPListener và UDPClient

11.1. TCPClient

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpClientSample
{
	public static void Main()
	{
		byte[] data = new byte[1024];
		string input, stringData;
		TcpClient server;
		try
		{
			server = new TcpClient("127.0.0.1", 5000);
		}
		catch (SocketException)
		{
			Console.WriteLine("Khong the ket noi den server");
			return;
		}
		NetworkStream ns = server.GetStream();
		int recv = ns.Read(data, 0, data.Length);
		stringData = Encoding.ASCII.GetString(data, 0, recv);
		Console.WriteLine(stringData);
		while (true)
		{
			input = Console.ReadLine();
			if (input == "exit")
				break;
			ns.Write(Encoding.ASCII.GetBytes(input), 0,
			input.Length);
			ns.Flush();
			data = new byte[1024];
			recv = ns.Read(data, 0, data.Length);
			stringData = Encoding.ASCII.GetString(data, 0, recv);
			Console.WriteLine(stringData);
		}
		Console.WriteLine("Dang ngat ket noi voi server...");
		ns.Close();
		server.Close();
	}
}
```

11.2. TCPListener

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class TcpListenerSample
{
	public static void Main()
	{
		int recv;
		byte[] data = new byte[1024];
		TcpListener newsock = new TcpListener(5000);
		newsock.Start();
		Console.WriteLine("Dan cho client ket noi den...");
		TcpClient client = newsock.AcceptTcpClient();
		NetworkStream ns = client.GetStream();
		string welcome = "Hello Client";
		data = Encoding.ASCII.GetBytes(welcome);
		ns.Write(data, 0, data.Length);
		while (true)
		{
			data = new byte[1024];
			recv = ns.Read(data, 0, data.Length);
			if (recv == 0)
				break;
			Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
			ns.Write(data, 0, recv);
		}
		ns.Close();
		client.Close();
		newsock.Stop();
	}
}
```

11.3. UDPClient

UDPClient phía server:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpSrvrSample
{
	public static void Main()
	{
		byte[] data = new byte[1024];
		IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 5000);
		UdpClient newsock = new UdpClient(ipep);
		Console.WriteLine("Dang cho client ket noi den...");
		IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
		data = newsock.Receive(ref sender);
		Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
		Console.WriteLine(Encoding.ASCII.GetString(data, 0, data.Length));
		string welcome = "Xin chao client";
		data = Encoding.ASCII.GetBytes(welcome);
		newsock.Send(data, data.Length, sender);
		while (true)
		{
			data = newsock.Receive(ref sender);
```

```
                    Console.WriteLine(Encoding.ASCII.GetString(data, 0,
data.Length));
                    newsock.Send(data, data.Length, sender);
                }
        }
}
```

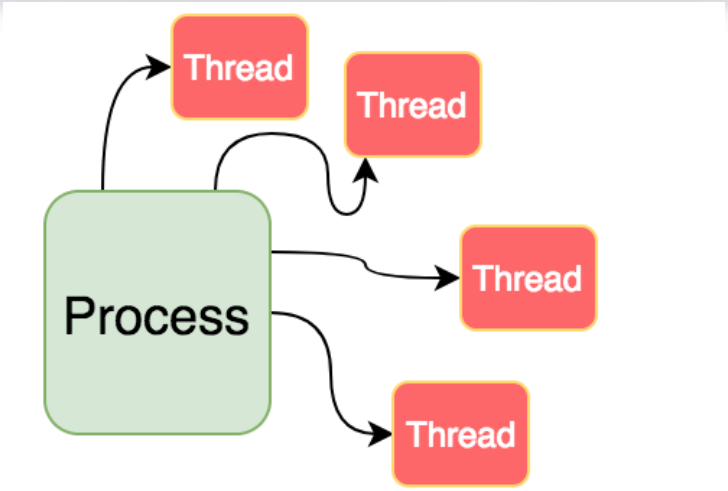
UDPClient phía client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class UdpClientSample
{
        public static void Main()
        {
                byte[] data = new byte[1024];
                string input, stringData;
                UdpClient server = new UdpClient("127.0.0.1", 5000);
                IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
                string welcome = "Xin chao server";
                data = Encoding.ASCII.GetBytes(welcome);
                server.Send(data, data.Length);
                data = server.Receive(ref sender);
                Console.WriteLine("Thong diep duoc nhan tu {0}:", sender.ToString());
                stringData = Encoding.ASCII.GetString(data, 0, data.Length);
                Console.WriteLine(stringData);
                while (true)
                {
                        input = Console.ReadLine();
                        if (input == "exit")
                                break;
                        server.Send(Encoding.ASCII.GetBytes(input), input.Length);
                        data = server.Receive(ref sender);
                        stringData = Encoding.ASCII.GetString(data, 0, data.Length);
                        Console.WriteLine(stringData);
                }
                Console.WriteLine("Dang dong client");
                server.Close();
        }
}
```

12. Đa nhiệm tiểu trình

12.1. Khái niệm

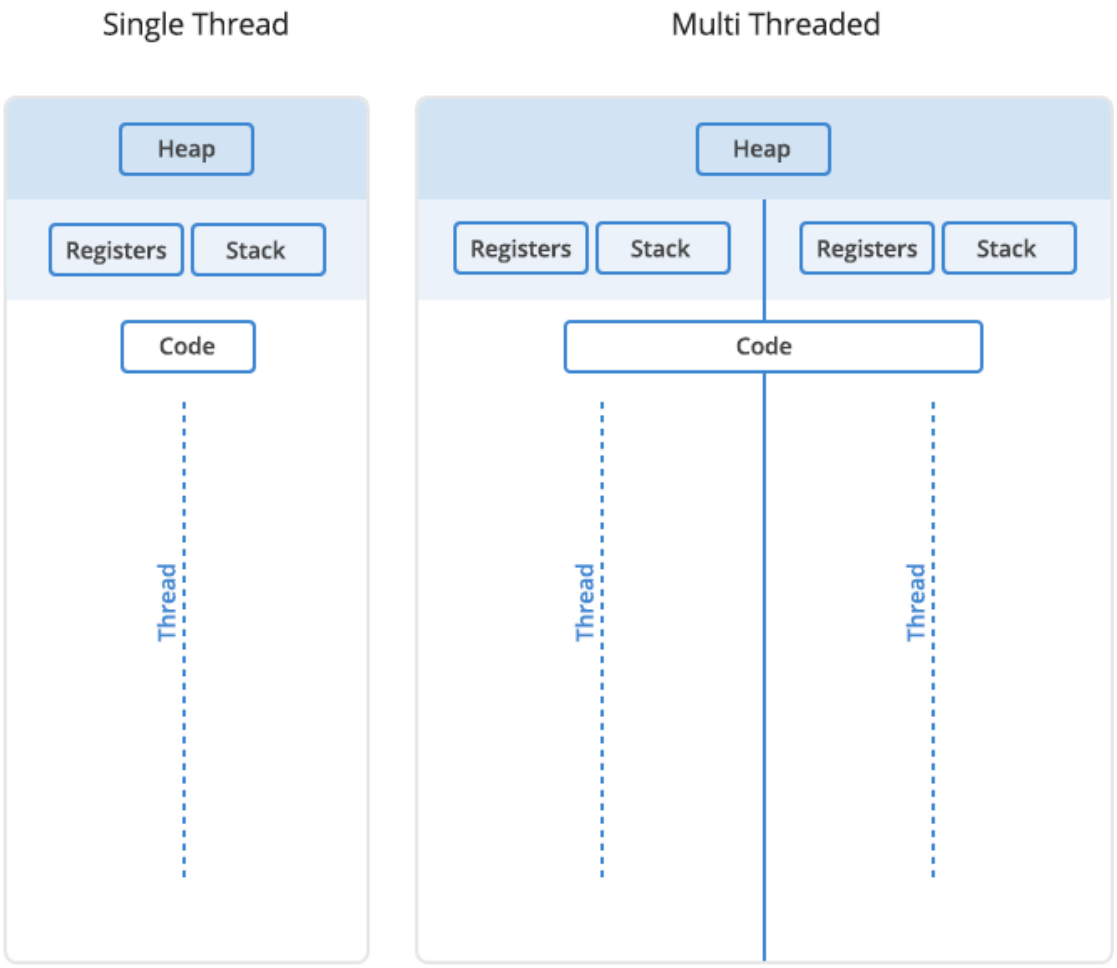
Tiến trình là một thể hiện của chương trình đang hoạt động. Một tiến trình luôn sở hữu một không gian địa chỉ có kích thước 4GB chứa mã chương trình, các dữ liệu, sở hữu tài nguyên của hệ thống như tập tin, đối tượng đồng bộ hóa....



Mỗi tiến trình khi mới tạo lập đều chỉ có một tiểu trình chính nhưng sau đó có thể tạo lập nhiều tiến trình khác.

Tiểu trình là một thành phần đơn vị của tiến trình có thể thực hiện các chỉ thị ứng với một đoạn mã nào đó của chương trình.

Hệ điều hành Windows cho phép các tiểu trình hoạt động độc lập và tổ chức điều phối (lập lịch tiến trình) CPU để các tiểu trình hoạt động đồng thời. Hệ điều hành phân chia thời gian sử dụng CPU cho mỗi tiến trình rất mịn theo kiểu xoay vòng. Mỗi tiểu trình có thể có 1 trong 3 trạng thái: Running, Ready, Blocked.



12.2. Tạo tiểu trình bất đồng bộ

Khi cho gọi một phương thức,thường thực hiện một cách đồng bộ; nghĩa là mã lệnh thực hiện lời gọi phải đi vào trạng thái dừng (block) cho đến khi phương thức được thực hiện xong.

Trong một số trường hợp, cần thực thi phương thức một cách bất đồng bộ; nghĩa là cho thực thi phương thức này trong một tiểu trình riêng trong khi vẫn tiếp tục thực hiện các công việc khác. Sau khi phương thức đã hoàn tất, cần lấy trị trả về của nó.

Có bốn kỹ thuật dùng để xác định một phương thức thực thi bất đồng bộ đã kết thúc hay chưa:

- **Blocking:** dừng quá trình thực thi của tiểu trình hiện hành cho đến khi phương thức thực thi bất đồng bộ kết thúc. Điều này rất giống với sự thực thi đồng bộ. Tuy nhiên, nếu linh hoạt chọn thời điểm chính xác để đưa mã lệnh vào trạng thái

dừng (block) thì vẫn còn cơ hội thực hiện thêm một số việc trước khi mã lệnh đi vào trạng thái này.

- **Polling:** lặp đi lặp lại việc kiểm tra trạng thái của phương thức thực thi bất đồng bộ để xác định nó kết thúc hay chưa. Đây là một kỹ thuật rất đơn giản, nhưng nếu xét về mặt xử lý thì không được hiệu quả. Nên tránh các vòng lặp chặt làm lãng phí thời gian của bộ xử lý; tốt nhất là nên đặt tiểu trình thực hiện polling vào trạng thái nghỉ (sleep) trong một khoảng thời gian bằng cách sử dụng Thread.Sleep giữa các lần kiểm tra trạng thái. Bởi kỹ thuật polling đòi hỏi phải duy trì một vòng lặp nên hoạt động của tiểu trình chờ sẽ bị giới hạn, tuy nhiên có thể dễ dàng cập nhật tiến độ công việc.
- **Waiting:** sử dụng một đối tượng dẫn xuất từ lớp System.Threading.WaitHandle để báo hiệu khi phương thức thực thi bất đồng bộ kết thúc. Waiting là một cải tiến của kỹ thuật polling, nó cho phépchờ nhiều phương thức thực thi bất đồng bộ kết thúc. Có thể chỉ định các giá trị time-out cho phép tiểu trình thực hiện waiting dừng lại nếu phương thức thực thi bất đồng bộ đã diễn ra quá lâu, hoặc muốn cập nhật định kỳ bộ chỉ trạng thái.
- **Callback:** Callback là một phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Mã lệnh thực hiện lời gọi không cần thực hiện bất kỳ thao tác kiểm tra nào, nhưng vẫn có thể tiếp tục thực hiện các công việc khác. Callback rất linh hoạt nhưng cũng rất phức tạp, đặc biệt khi có nhiều phương thức thực thi bất đồng bộ chạy đồng thời nhưng sử dụng cùng một callback. Trong những trường hợp như thế, phải sử dụng các đối tượng trạng thái thích hợp để so trùng các phương thức đã hoàn tất với các phương thức đã khởi tạo.

Ví dụ chương trình đa tiểu trình:

```
using System;
using System.Threading;
public class ThreadExample
{
	// Các biến giữ thông tin trạng thái.
	private int iterations;
	private string message;
	private int delay;
	public ThreadExample(int iterations, string message, int delay)
	{
		this.iterations = iterations;
		this.message = message;
		this.delay = delay;
```

```
        }

        public void Start()
        {
                // Tạo một thể hiện ủy nhiệm ThreadStart
                // tham chiếu đến DisplayMessage.
                ThreadStart method = new ThreadStart(this.DisplayMessage);
                // Tạo một đối tượng Thread và truyền thể hiện ủy nhiệm
                // ThreadStart cho phương thức khởi dựng của nó.
                Thread thread = new Thread(method);
                Console.WriteLine("{0} : Starting new thread.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
                // Khởi chạy tiểu trình mới.
                thread.Start();
        }

        private void DisplayMessage()
        {
                // Hiển thị thông báo ra cửa sổ Console với số lần
                // được chỉ định (iterations), nghỉ giữa mỗi thông báo
                // một khoảng thời gian được chỉ định (delay).
                for (int count = 0; count < iterations; count++)
                {
                        Console.WriteLine("{0} : {1}",
                        DateTime.Now.ToString("HH:mm:ss.ffff"), message);
                        Thread.Sleep(delay);
                }
        }

        public static void Main()
        {
                // Tạo một đối tượng ThreadExample.
                ThreadExample example = new
                ThreadExample(5, "A thread example.", 500);
                // Khởi chạy đối tượng ThreadExample.
                example.Start();
                // Tiếp tục thực hiện công việc khác.
                for (int count = 0; count < 13; count++)
                {
                        Console.WriteLine("{0} : Continue processing...",
                        DateTime.Now.ToString("HH:mm:ss.ffff"));
                        Thread.Sleep(200);
                }
                // Nhấn Enter để kết thúc.
                Console.WriteLine("Main method complete. Press Enter.");
                Console.ReadLine();
        }
}
```

Ví dụ tạo tiến trình mới:

```
using System;
using System.Diagnostics;
public class StartProcessExample
{
        public static void Main()
        {
                // Tạo một đối tượng ProcessStartInfo và cấu hình cho nó
                // với các thông tin cần thiết để chạy tiến trình mới.
                ProcessStartInfo startInfo = new ProcessStartInfo();
                startInfo.FileName = "notepad.exe";
                startInfo.Arguments = "file.txt";
```

```
                startInfo.WorkingDirectory = "C:\\Temp";
                startInfo.WindowStyle = ProcessWindowStyle.Maximized;
                startInfo.ErrorDialog = true;
                // Tạo một đối tượng Process mới.
                using (Process process = new Process())
                {
                        // Gán ProcessStartInfo vào Process.
                        process.StartInfo = startInfo;
                        try
                        {
                                // Khởi chạy tiến trình mới.
                                process.Start();
                                // Đợi tiến trình mới kết thúc trước khi thoát.
                                Console.WriteLine("Waiting 30 seconds for process to
finish.");
                                process.WaitForExit(30000);
                        }
                        catch (Exception ex)
                        {
                                Console.WriteLine("Could not start process.");
                                Console.WriteLine(ex);
                        }
                }
                // Nhấn Enter để kết thúc.
                Console.WriteLine("Main method complete. Press Enter.");
                Console.ReadLine();
        }
}
```

13. Đồng bộ hóa

Trong các hệ điều hành đa nhiệm cho phép nhiều công việc được thực hiện đồng thời. Việc tồn tại cùng lúc nhiều tiểu trình trong môi trường có thể dẫn đến sự tranh chấp, ngăn cản hoạt động lẫn nhau giữa các tiểu trình.

Ví dụ: Với một tiến trình mới đầu tạo lập 4 tiểu trình cùng có nội dung xử lý đồng nhất. Mỗi tiểu trình sẽ tăng giá trị của biến toàncục Count lên 250.000 lần. Do vậy Count sẽ tăng lên 1.000.000 lần trong toàn bộ tiến trình.

Để hạn chế các trình trạng tranh chấp tài nguyên cần có cơ chế điều khiển các tiểu trình truy xuất tài nguyên một cách tuần tự. đó chính là thực hiện đồng bộ hóa các tiểu trình. Khi một tiểu trình truy xuất đến một tài nguyên dùng chung , cần chú ý thực hiện đồng bộ hóa việc truy xuất tài nguyên để tránh xảy ra tranh chấp. Các cơ chế đồng bộ hóa đều dựa trên ý tưởng chỉ cho phép một tiểu trình được truy xuất tài nguyên khi thỏa điều kiện không có tranh chấp trên đó. Những tiểu trình không hội đủ điều kiện để sử dụng tài nguyên thì được hệ điều hành đặt vào trạng thái chờ (Không chiếm CPU), và hệ điều hành sẽ luôn kiểm sóat tình trạng truy xuất tài nguyên của tiểu trình khác để có thể giải phóng kịp thời các tiểu trình đang chờ vào thời điểm thích hợp.

13.1. Semaphore

Ví dụ: Tạo ra 10 tiểu trình nhưng tại một thời điểm chỉ có 3 tiểu trình truy cập tài nguyên chung:

```
class SemaphoreTest
{
	static Semaphore s = new Semaphore(3, 3);
	// Available=3; Capacity=3
	static void Main()
	{
		for (int i = 0; i < 10; i++)
		{
			Thead thread = new Thread(new ThreadStart(Go));
			thread.Start();
		}
	}

	static void Go()
	{
		while (true)
		{
			s.WaitOne();
			Thread.Sleep(100);
			// Only 3 threads can get here at once
			s.Release();
		}
	}
}
```

13.2. Mutex

Mutex cung cấp một cơ chế để đồng bộ hóa quá trình thực thi của các tiểu trình vượt qua biên tiến trình. Một Mutex là signaled khi nó không thuộc sở hữu của bất kỳ tiểu trình nào. Một tiểu trình giành quyền sở hữu Mutex lúc khởi dựng hoặc sử dụng một trong các phương thức được liệt kê ở trên.

Quyền sở hữu Mutex được giải phóng bằng cách gọi phương thức Mutex.ReleaseMutex (ra hiệu Mutex và cho phép một tiểu trình khác thu lấy quyền sở hữu này).

Ví dụ dưới đây sử dụng một Mutex có tên là MutexExample để bảo đảm chỉ một thể hiện của ví dụ có thể thực thi:

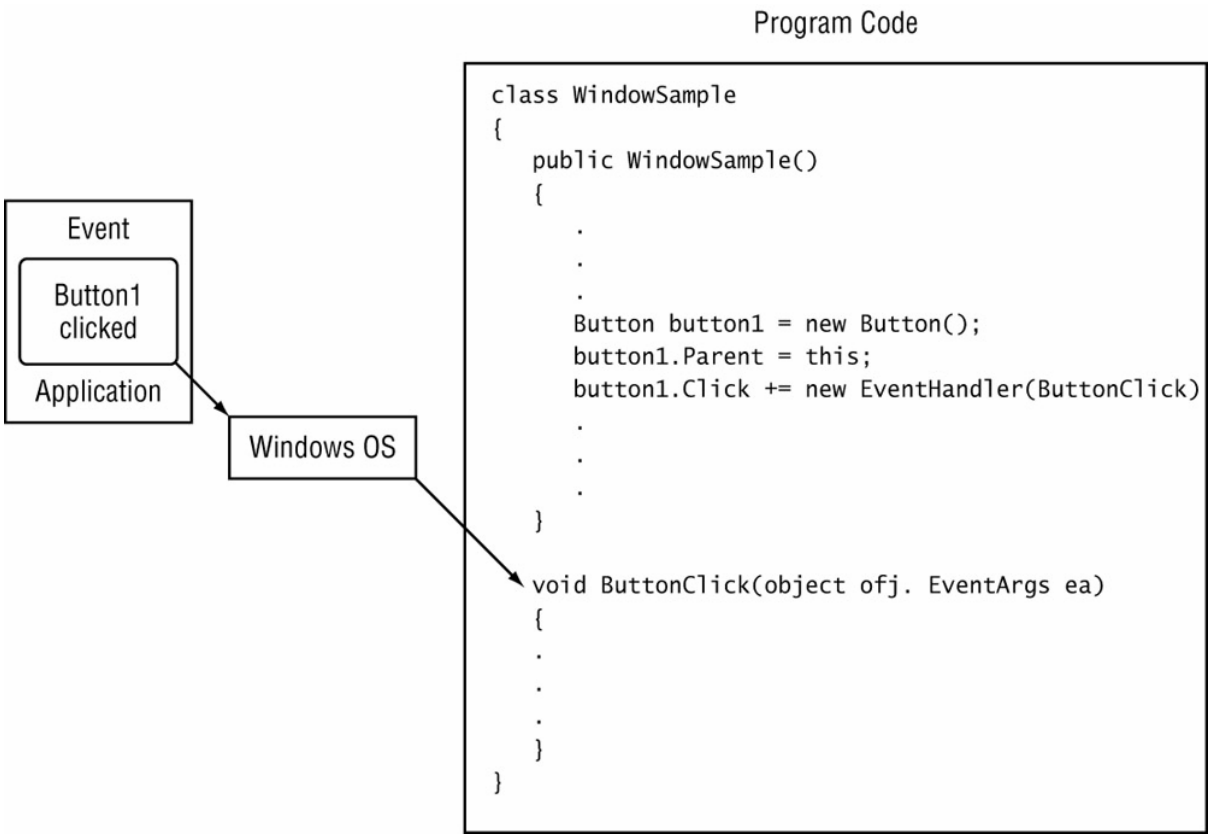
```
using System;
using System.Threading;

public class MutexExample
{
	public static void Main()
	{
		// Giá trị luận lý cho biết ứng dụng này
		// có quyền sở hữu Mutex hay không.
		bool ownsMutex;
		// Tạo và lấy quyền sở hữu một Mutex có tên là MutexExample.
		using (Mutex mutex = new Mutex(true, "MutexExample", out ownsMutex))
		{
```

```
                        // Nếu ứng dụng sở hữu Mutex, nó có thể tiếp tục thực thi;
                        // nếu không, ứng dụng sẽ thoát.
                        if (ownsMutex)
                        {
                                Console.WriteLine("This application currently owns the
mutex named MutexExample.Additional instances of this application will not run until
       you release the mutex by pressing Enter.");
                                Console.ReadLine();
                                // Giải phóng
                                Mutex.mutex.ReleaseMutex();
                        }
                        else
                        {
                                Console.WriteLine("Another instance of this application
already owns the mutex named MutexExample.This instance of the application will
terminate.");
                        }
                }
                // Nhấn Enter để kết thúc.
                Console.WriteLine("Main method complete. Press Enter.");
                Console.ReadLine();
        }
}
```

14. Lập trình Socket bất đồng bộ

Trong lập trình sự kiện trong Windows, mỗi khi một sự kiện xảy ra, một phương thức được gọi để thực thi dựa trên sự kiện đó như trong hình dưới đây:



Trong các chương trước, chúng ta đã lập trình Socket trong chế độ blocking. Socket ở chế độ blocking sẽ chờ mãi mãi cho đến khi hoàn thành nhiệm vụ của nó. Trong khi nó bị blocking thì các chức năng khác của chương trình không thực hiện được.

Khi lập trình Windows thì lúc gọi một phương thức bị blocking thì toàn bộ chương trình sẽ đứng lại và không thực hiện các chức năng khác được. Do đó việc lập trình bất đồng bộ là cần thiết để cho chương trình khỏi bị đứng.

Đối tượng Socket có nhiều phương thức sử dụng lớp AsyncCallback để gọi các phương thức khác khi các chức năng mạng hoàn tất. Nó cho phép dùng tiếp tục xử lý các sự kiện khác trong khi chờ cho các chức năng mạng hoàn thành công việc của nó.