Internet Relay Chat Protocol - Tìm hiểu và ứng dụng

Giới Thiệu Đề Tài

Đề Tài: Internet Relay Chat Protocol tìm hiểu và ứng dụng *do thầy* Mai Văn Cường *hướng dẫn, luận văn sau đây giới thiệu một ứng dụng hay nói đúng hơn là một chương trình cho phép người sử dụng chat với nhau trên Internet. Chương trình hoạt động theo đúng giao thức chuẩn do ủy ban IAB về các giao thức chuẩn (IAB Official Protocol Standards) quy định, quy định này được định nghĩa trong bộ RFC 1459, 2810, 2811, 2812, 2813.*

Luận văn gồm 3 phần chính

- *Phần I trong luận văn là tìm hiểu Internet Relay Chat Protocol (IRC protocol)*

Internet Relay Chat Protocol *là giao thức chuẩn cho các chương trình chat hiện tại, các chương trình muốn thực hiện được trên Internet phải tuân theo giao thức này. Nội dung chính của giao thức này là các message được gởi và nhận giữa client và server.*

- *Phần II các khái niệm về mạng và mô hình socket để tạo giao tiếp phục vụ cho việc truyền nhận dữ liệu, chương trình chat application chủ yếu sử dụng những phương thức của socket để hoạt động.*

- *Phần III một số vấn đề đa xử lý và đa luồng giúp cho chương trình hoạt động hiệu quả hơn.*

- *Phần IV các yêu cầu chức năng, phi chức năng và kiến trúc của chương trình*

- *Phần cuối là hướng phát triển đề tài và tài liệu tham khảo, phần phụ lục dành để tham khảo giá trị trả về trong bộ giao thức*

Phần I:
Tìm Hiểu Internet Relay Chat (IRC) protocol

- ✓ Giới thiệu tổng quan về IRC
- ✓ Chương I: Một số khái niệm cơ bản
- ✓ Chương II: Những quy định trong IRC
- ✓Chương III: Chi tiết cho từng message

Giới Thiệu Tổng Quan Về IRC

Hiện nay trên Internet có nhiều loại dịch vụ, mỗi dịch vụ cung cấp cho chúng ta một tiện ích khác nhau, trong đó có dịch vụ chat. Đây là loại dịch vụ cho phép mọi người trên khắp hành tinh có thể gặp gỡ, trao đổi thông tin với nhau mà không cần phải gặp nhau trực tiếp. Dịch vụ này rất phát triển, có đến hàng triệu người trên thế giới đang sử dụng dịch vụ này. Vì vậy vấn đề đặt ra là phải có một quy định chung cho hệ thống mạng IRC(Internet Relay Chat). Ủy Ban IAB về các giao thức chuẩn (IAB Official Protocol Standards) đã đưa ra một giao thức chuẩn (Standard protocol) dùng cho tất cả các chương trình chat đang tồn tại. Đó là IRC (Internet Relay Chat) protocol được định nghĩa trong RFC(Request For Comment) 1459, 2810, 1324, 2811, 2813.

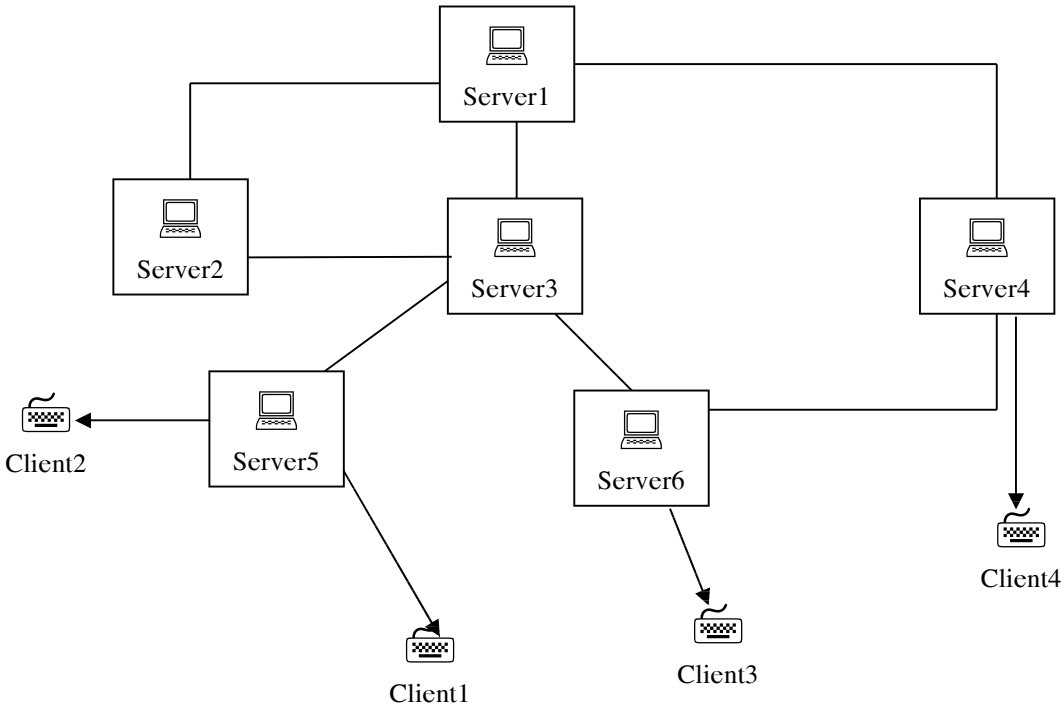
IRC Protocol được đưa ra vào năm 1980, tiền thân của nó, dùng cho các thành viên trong mạng BBS trao đổi thông tin với nhau, dần dần được cải tiến và trở thành giao thức chuẩn cho các chương trình IRC. Quy mô của IRC protocol là trên toàn cầu, gồm có 2 thành phần Client và Server.

Hiện nay IRC Protocol được xây dựng trên họ giao thức mạng phổ biến nhất là* TCP/IP (TCP/IP Net Work Protocol) *lý do việc sử dụng họ giao thức này là tính chính xác, tin cậy, phổ biến, rất thích cho các cuộc thảo luận từ xa.

IRC Protocol dùng mô hình client – server, vì thế chúng ta có thể chạy nhiều máy trên môi trường phân tán (distributed enviroment). Trong đó máy đóng vai trò là server cung cấp một điểm tập trung (central point) cho các client kết nối đến, và đồng thời thực hiện quá trình truyền nhận message từ các client này đến các client khác.

Chương I: Một số khái niệm cơ bản.

1. Servers:



Hình 1: *mô hình hệ thống mạng IRC*

Server được xem là xương sống của mạng IRC, mỗi server là một tâm điểm trong hệ thống các server, chúng cho phép client và server khác kết nối vào. Những server này được kết nối theo biểu đồ hình cây (spanning tree).

Ví dụ: client 1 muốn trao đổi thông tin với client 3 chúng phải thông qua server5, server6, server 3.

2. Clients:

Client là một máy tính mà nó được kết nối đến server và máy tính đó không phải là server (Xem :Hình1.1)

Client là thiết bị đầu cuối nó không chuyển tiếp message cho bất cứ máy tính nào khác

Mỗi client được phân biệt với nhau thông qua Nickname (Nickname là chuỗi có giá trị tối đa 9 ký tự). Server dùng Nickname để quản lý các client.

Khi có sự tham gia của một client vào hệ thống, tất cả các server phải có thông tin về client đó như là tên client (Hostname), tên server mà nó kết nối đến v.v ...

2.1. Operators:

Để có thể quản lý số lượng user tham gia trên mạng (IRC network) người ta xây dựng một nhóm user gọi là "client operator" nhóm này có đầy đủ mọi quyền hạn trên mạng (IRC network). Mặc dù quyền hạn (cấp cho client operation) có thể được xem là "nguy hiểm", nhóm

"client operator" có thể thực hiện các tác vụ như hủy bỏ kết nối (disconnecting) của một client nào đó hoặc tái kết nối (reconnecting) đến server nào đó. Những tác vụ này có thể thực hiện khi có một server bị hỏng hay khi có sự cố về đường truyền và nhiều nguyên nhân khác. Chính vì khả năng đó nên người ta cho rằng nhóm "client operators" có thể nguy hiểm. vì có thể nhóm client này hủy bỏ kết nối của một client khác mà không có lý do hợp lý, hay mục đích chính đáng.

3. Channels:

Channel là tên nhóm một hay nhiều client, mà những client này sẽ cùng được nhận các message gởi đến channel đó, nói cách khác là các client thuộc về một channel sẽ nhận được message gởi đến channel đó. Những client trong cùng một channel mới có thể nhận thấy nhau.

Một channel được ngầm tạo ra khi có client đầu tiên tham gia và kết thúc khi client cuối cùng ngưng kết nối. Trong khi channel đang tồn tại thì client có thể tham gia vào channel đó bằng cách dùng tên channel.

Tên của channel có thể lên đến 200 ký tự và bắt đầu bằng ký tự '&' hoặc ký tự '#', Tên channel không có ký tự khoảng trắng (' '), Ctr+G(^G or ASCII 7), dấu phẩy (',').

Để có thể tạo ra một channel hay là tham gia vào một channel có sẵn client phải gởi JOIN message để tham gia vào channel đó.

3.1.Channels operation :

Channel oprerator còn được gọi là "chop" hoặc "chanop". Khi có một user tạo ra channel thì mặt nhiên user đó trở thành channel operator, là người sở hữu channel user (channel operator) có đầy đủ mọi quyền hạn trên channel đó. Để quản lý các client channel operator có thể thực hiện quyền của mình như :

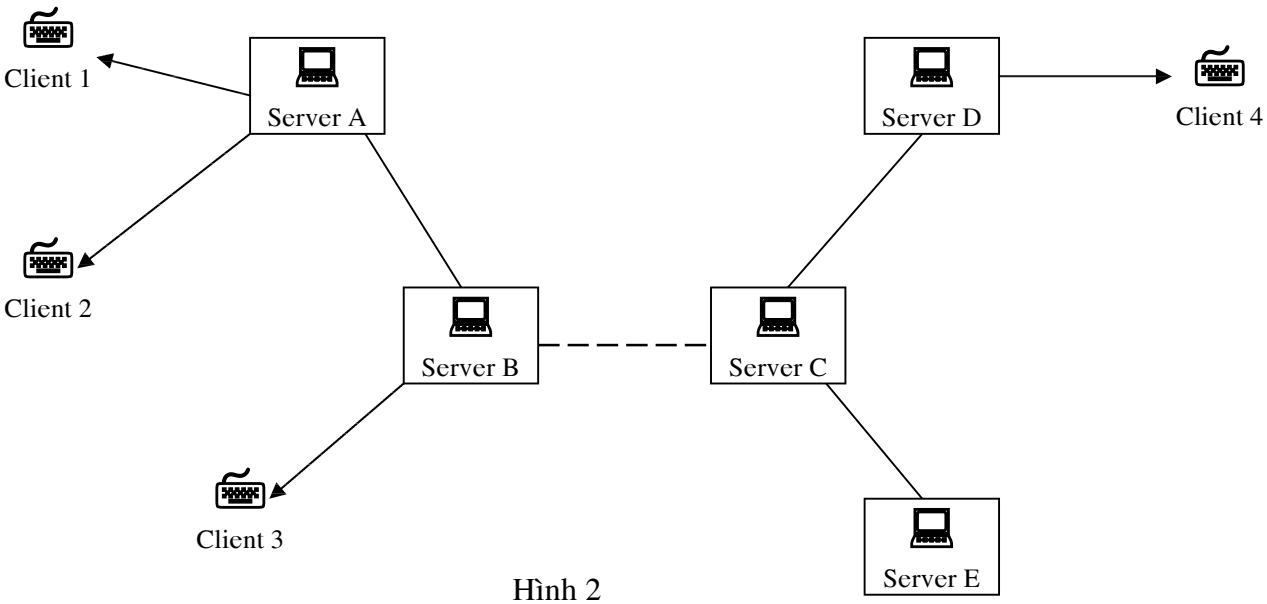
- ⇨ KICH - đẩy một client ra khỏi channel
- ⇨ MODE - thay đổi mode của channel
- ⇨ INVITE - gọi một client tham gia vào channel mà nó đang ở trạng thái invite-only (mode i+).
- ⇨ TOPIC - thay đổi topic channel, channel này đang ở trạng thái +t(mode +t).

Ngoài ra channel operator có thể cấp quyền cho client khác hay nhường quyền channel operator lại. Tuy nhiên quyền hạn này không được chính xác vì những nguyên nhân đã được trình bài phần trên.

Một channel operation thì được nhận dạng bởi ký tự bắt đầu "@" tiếp theo là nickname (nick name của user tạo ra channel đó). Ví dụ @HappyMan

4. Các khái niệm về truyền thông:

4.1. Truyền thông một – một (one – to – one communication)



Hình 2

Truyền thông 1-1 phục vụ cho việc chuyển thông điệp giữa hai client. Chuyển thông điệp này mang ý nghĩa là riêng biệt giữa 2 client nhưng thật ra vẫn phải thông qua các server mà những client đó kết nối đến.

Ví dụ : xem hình 1.2 : client 1 giao tiếp với client 2 thông qua Server A.
Message từ client 1 đến client 3 phải được chuyển qua ServerA và ServerB trong khi đó các Server và client còn lại không được nhận message.

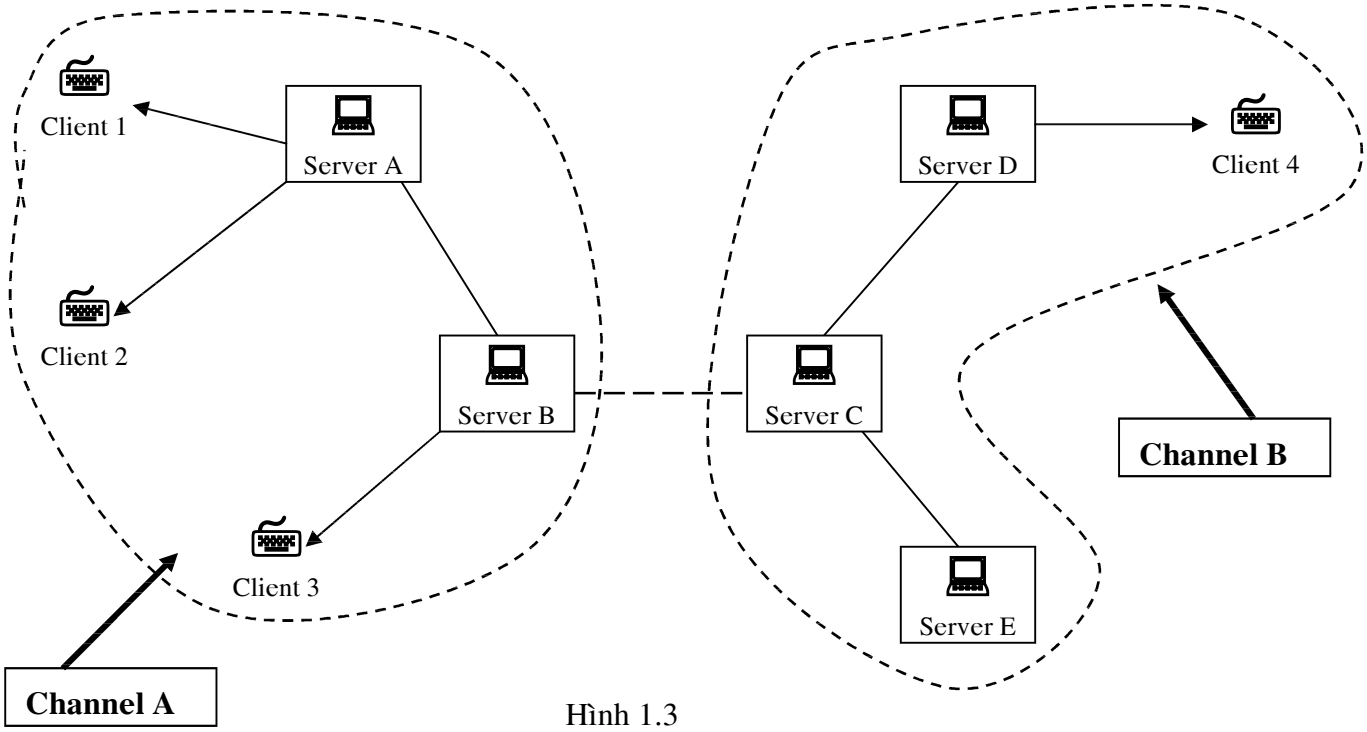
4.2. Truyền thông một – nhiều (one – to – many communication)

Mục đích của loại truyền thông này là cung cấp cho các cuộc hội thảo, thảo luận v.v... Loại truyền thông này dựa trên nhiều cách.

4.2.1. Dựa trên danh sách (one – to – List communication)

Đây là cách truyền thông điệp kém hiệu quả nhất trong truyền thông 1-n. Danh sách này có thể là các client, các server, hoặc là các mask. Server khi nhận được thông điệp có đích đến là một danh sách, nó sẽ chuyển thông điệp này đến tất cả các đích. Cách chuyển này không hiệu quả ở chỗ nó không kiểm tra các đích đến có cùng trên một con đường không, và như vậy có khả năng sẽ có nhiều thông điệp giống nhau cùng chuyển trên một con đường.

4.2.2.Dựa vào channel (one – to – group communication)



Hình 1.3

Một channel được tạo ra "động"(dynamic). Vì khi có một user tham gia, nó được tạo ra và bị hủy khi user cuối rời khỏi. Trên thực tế, message gởi đến channel thì chỉ gởi duy nhất một lần.

❋ Nếu có nhiều user trên cùng server mà những user này thuộc về một channel, message gởi đến channel sau đó được server chuyển đến các thành viên do nó quản lý.

❋ Nếu có nhiều user trên những server khác nhau và chúng trong cùng một channel, message gởi đến channel sẽ được gởi đến các server có user kết nối đến, việc còn lại do server chuyển message đó cho các thành viên của nó.

Vídụ: Xem hình Hình1.3 ta thấy client 1, 2, 3 trong channelA.

Khi có message vào channelA. Nếu message đến client 1, 2 thì nó chuyển đến cho serverA, message đến client 3 và chuyển đến cho server B, nếu message chuyển cho cả channelA thì nó sẽ chuyển đến cả hai server A và B.

Đối với loại PRIVMSG message khi vào channel, nó được chuyển đến server (có user nhận), công việc còn lại là server chuyển cho user đó.

4.2.3Truyền thông đến - host /server mask

Truyền thông điệp đến host/server mask là cách để gửi thông điệp đến cho nhiều người dùng có cùng một vài thông tin đặc điểm về host và server nào đó.

Cách truyền thông này gần giống như channel.

Ví dụ: có 3 người dùng với host như sau : abc.pacific.au, abd.gulu.fi, def.gulu.au. Khi ta đề cập đến các người dùng có host mask *.au tức là nói đến abc.pacific.au, def.gulu.au

4.3. Truyền thông một - tất cả(one to all Communication)

Có thể xem kiểu truyền thông này giống như phát quảng bá (broadcast). Trong đó, message được gởi đến tất cả các client và Server trên mạng và chúng sẽ tự tìm kiếm con đường trên mạng để đến tất cả các địa chỉ đích (client destination)

4.3.1.Client to Client

Khái niệm tương tự như trên. Message từ một client đến một client khác.

4.3.2.Client to Server

Hầu hết những tác vụ command mà chúng có khả năng làm thay đổi trạng thái hoạt động như là: Channel membership, channel mode, user status, v, v....phải được gởi đến server server để thực hiện, những giá trị này không được thay đổi bởi bất cứ client nào.

Chương II: Những quy định trong IRC

1. Dạng chung của thông điệp

Thông điệp thường có 2 dạng:

Các thông điệp xuất phát từ lệnh (thông điệp được phát sinh từ một lệnh): là thông điệp được client gửi lên server để yêu cầu một mục đích nào đó (ví dụ như hỏi thông tin một nick nào đó, hay hỏi thông tin các channel, hoặc chỉ đơn giản là muốn gửi thông điệp cần nói trên channel...). Các thông điệp này có thể không còn nguyên thủy như khi user gõ vào, mà chúng được gắn thêm prefix (địa chỉ host, nickname... của user gửi).

Thông điệp trả về: là thông điệp được server gửi về client để trả lời lại các yêu cầu của client.

Các thông điệp có hai dạng: thông điệp bằng chữ và thông điệp bằng số.

2. Message

Message là thông điệp từ client gởi cho server hoặc ngược lại, nếu trong message chứa lệnh (lệnh này sẽ được mô tả phần sau), thì những lệnh này sẽ được đáp lại bằng thông điệp phản hồi (reply message).

Mỗi message gồm có 3 phần chính: phần đầu còn gọi là tiếp đầu ngữ (prefix), tiếp theo đó là phần lệnh (command) và cuối cùng là danh sách đối số(parameters list), mỗi phần cách nhau bởi ký tự khoảng trắng(ASCII 0x20). Bắt đầu prefix là ký tự ":" (ASCII 0x3b) chính nhờ vào ký tự này mà server nhận biết chính xác phần lệnh(command), những cú pháp lệnh sẽ được mô tả chi tiết trong phần định dạng message(format message).

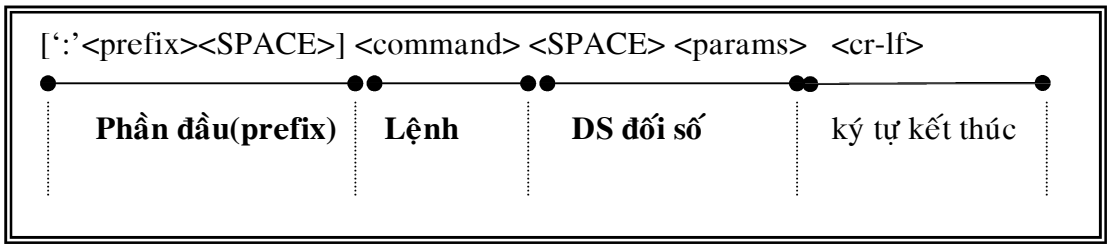
Nếu message từ client gởi đến mà không nhận thấy phần đầu (prefix) thì server xem như message đó được gởi từ client mà nó kết nối trực tiếp, vì thế các client muốn gởi message đến server mà nó kết nối trực tiếp không nên dùng phần prefix, nếu sử dụng cần phải sử dụng chính xác vì nếu prefix không đúng thì server không đáp lại mà nó im lặng.

Phần lệnh (command) là những thành phần thuộc về IRC(được mô tả chi tiết trong phần message detail), chiều dài lệnh và danh sách đối số cho phép là 510 ký tự , cuối mỗi message có ký tự kết thúc (CR-LF) như thế tổng chiều dài chúng là 512 ký tự.

2.1 Định dạng một message

Vấn đề đặt ra là làm sao có thể nhận dạng và phân tích ra từ những luồng tuần tự các message gởi đến, để giải quyết vấn đề này người ta lập ra quy định cho message.

Như đã mô tả ở phần trên, một message luôn có ký tự kết thúc nhờ vào ký tự này mà server có thể tách message ra khỏi luồng (stream). Sau khi tách ra chúng được phân tích thành 3 thành phần chính. chúng ta sẽ khảo sát chúng trong phần sau.bây giờ chúng ta phân tích cấu trúc tổng quát cho một message:



Giải thích cú pháp:

- Đối tượng nằm trong dấu [ ] có thể có hoặc không có.
- Đối tượng nằm trong dấu < > là bắt buột phải có.
- Toán tử ' | ' là có khả năng chọn một trong hai ví dụ: "<a> | <b>" có thể chọn <a> hoặc chọn <b>.
- Đối tượng nằm trong dấu {} có thể xuất hiện nhiều lần.

3 thành phần chính của message:

➢**Phần đầu(prefix)**

Phần prefix có thể không có, bắt đầu prefix phải có dấu ':' :

<prefix> ::=<servername > | <nick> ['!' <user>]['@' <host>]

Ở đây chúng ta có thể thấy prefix có thể là **servername** hoặc **nickname** ngoài ra còn có thể có thêm **username** hay **hostname** hoặc cả hai. Nhưng đối với username phải có ký tự '!' đứng trước, tương tự như thế hostname phải có ký tự '@'đứng đầu.

➢**Phần Lệnh(command)**

Phần lệnh có thể ở dạng chuỗi hoặc số :

<command> ::=<letter> {<letter>} | <number> <number> <number>

dễ dàng chúng ta có thể nhận thấy lệnh có thể do một hay nhiều ký tự hoặc giá trị số gồm 3 chữ số.

➢**Phần danh sách đối số(parameters list)**

<params> ::=<SPACE> [':' <trailing> | <middle> <params>]

Chúng ta thấy đối số có thể không xuất hiện, giải thích thêm về <trailing><middle> <trailing> là chuỗi ký tự đại diện trong đó không có ký tự NULL hoặc CR, LF. Chuỗi này được đặt giữa hai ký tự "* * "

<middle> cũng là chuỗi đại diện nhưng có sự phân biệt, ký tự đầu chuỗi không được là ký tự ':'(ASCII 0x3b).

Ví dụ: "* !* " đại diện cho tất cả các user.

"*@* " đại diện cho tất cả host name.

"*.edu " đại diện cho nhóm server có phần cuối là edu.

3. Giá trị số trả về (numeric replies)

Sau khi khảo sát về cấu trúc của một message, chúng ta được biết một message được nhận dạng như thế nào. Sau khi nhận được message, server sẽ phát ra message phản hồi (reply message). Reply Message được hiểu tương tự như là message, thật sự nó gồm 3 phần: sender prefix, giá trị số gồm 3chữ số và target.

- sender prefix là nickname của client gởi.
- giá trị số được mô tả phần sau.
- <taget> ::= <channel> | <user> '@' <servername> | <nick> | <mask> ["," <taget>] để nhận biết reply message đến client nào chúng dựa vào <target>.

Ở đây chúng ta có 2 loại message phản hồi:

error reply

normal reply

Lưu ý: chỉ có server mới có khả năng phát ra reply message.

Chương III: Chi tiết cho từng message(Message detail)

Đây là phần mô tả chính cho mỗi loại message, để sever và client có thể nhận biết chúng. Server sẽ nhận message và phân tích chúng, sau đó trả lại thông báo thích hợp. Nếu phân tích message mà gặp phải lỗi(error). Khi đó server phải có cơ chế thông báo cho client.

Một lỗi(error) sinh ra thường do những nguyên nhân sau:

- ☑ Sai đối số (incorrect parameter),
- ☑ Sai Lệnh(incorrect command),
- ☑ Sai địa chỉ đích Tên server
  NickName
  Channel name
- ☑ Sai vì vi phạm quyền hạn v.v...

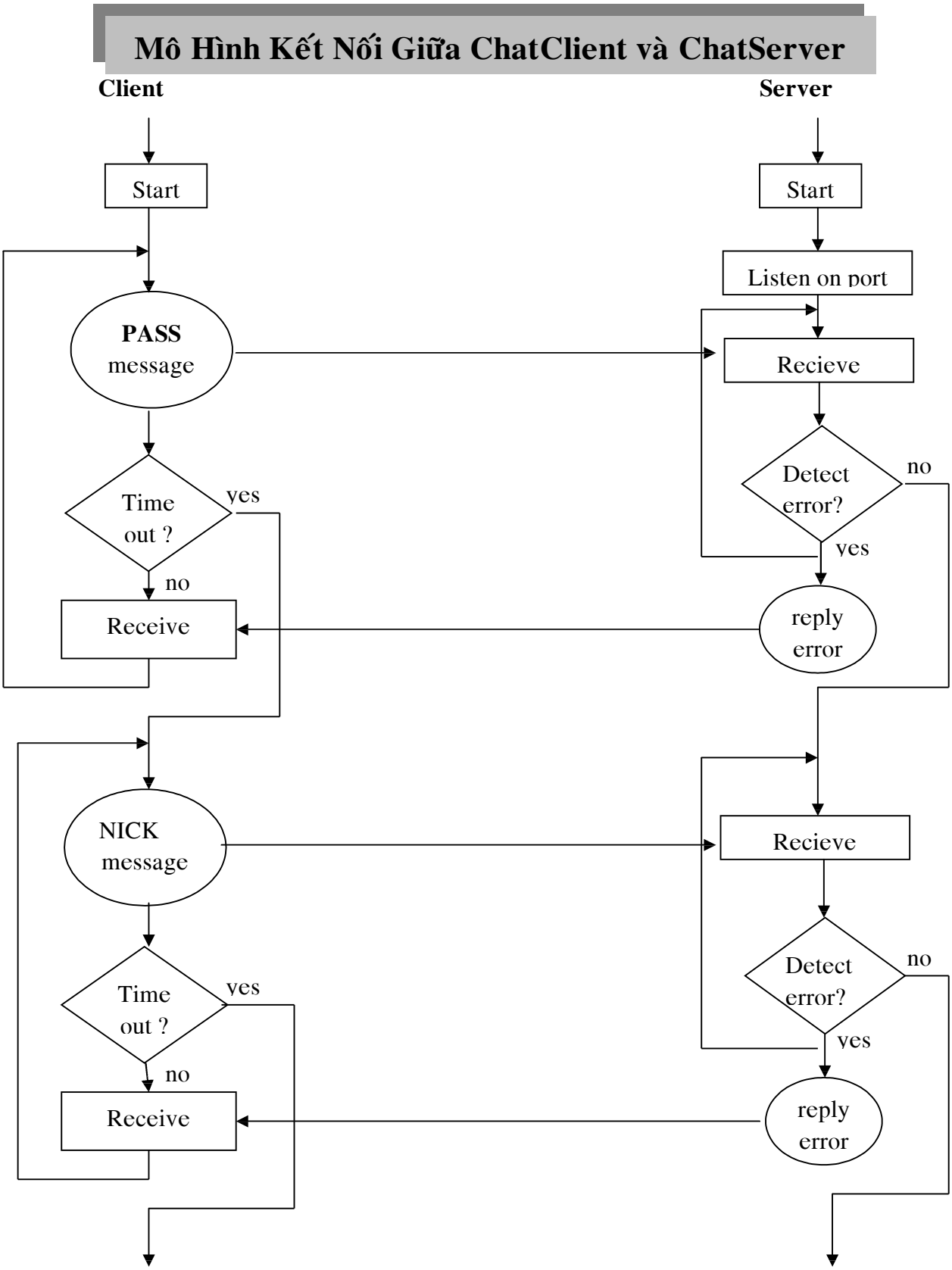
Cú pháp cho một command message

:Name COMMAND parameter list

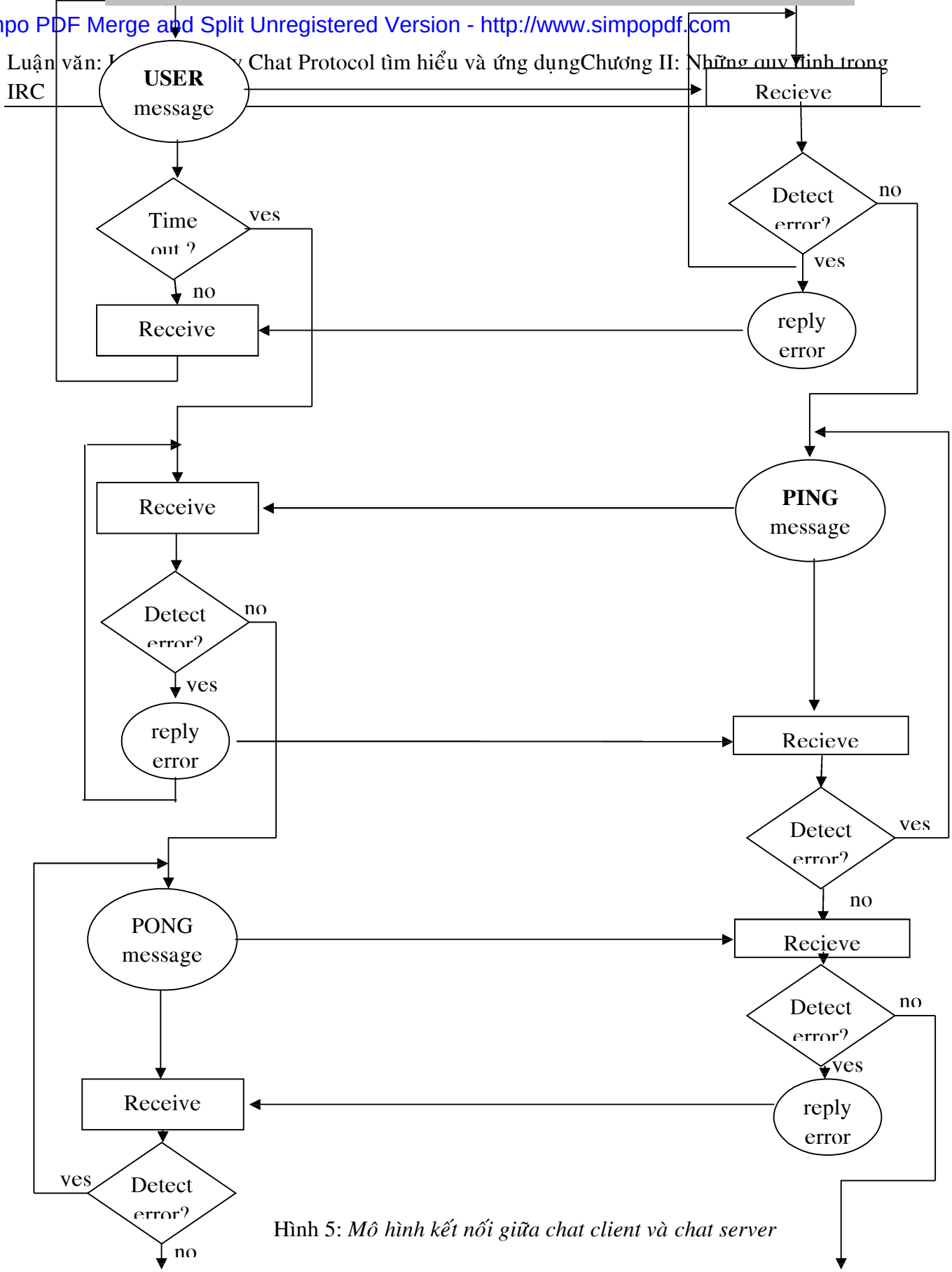
Chú ý: Đối với "**Name**", đó là tên của client gởi. Một server(từ xa) được client gởi message đến thì server sẽ căn cứ vào "**Name**", để có thể đáp lại yêu cầu ngược lại nếu client gởi đến server mà nó kết nối trực tiếp không cần.

1. Nhóm message đăng ký kết nối

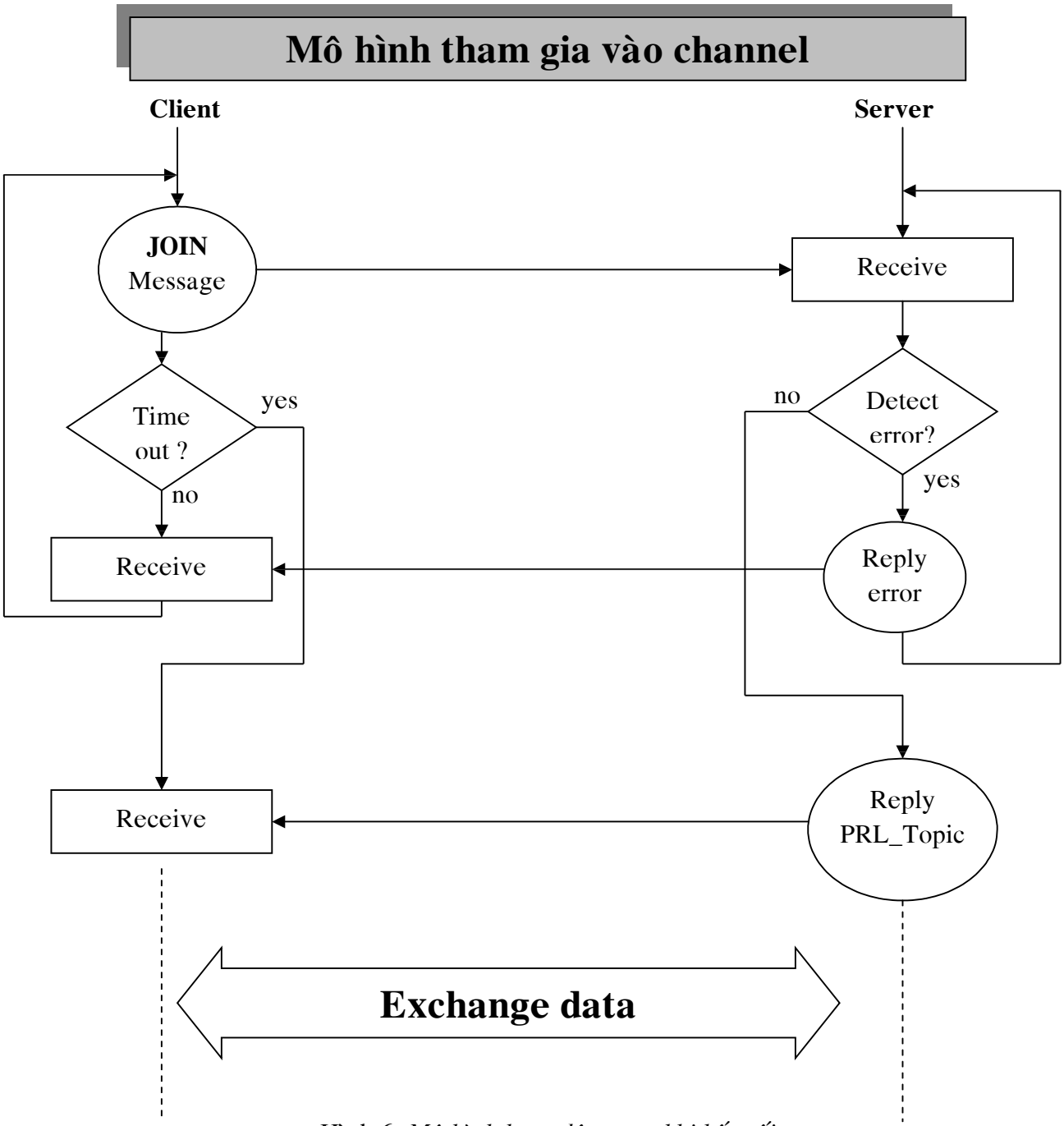
Nhóm message đăng ký kết nối chịu trách nhiệm kết nối với IRC server, giao thức kết nối được mô tả trong lưu đồ sau:



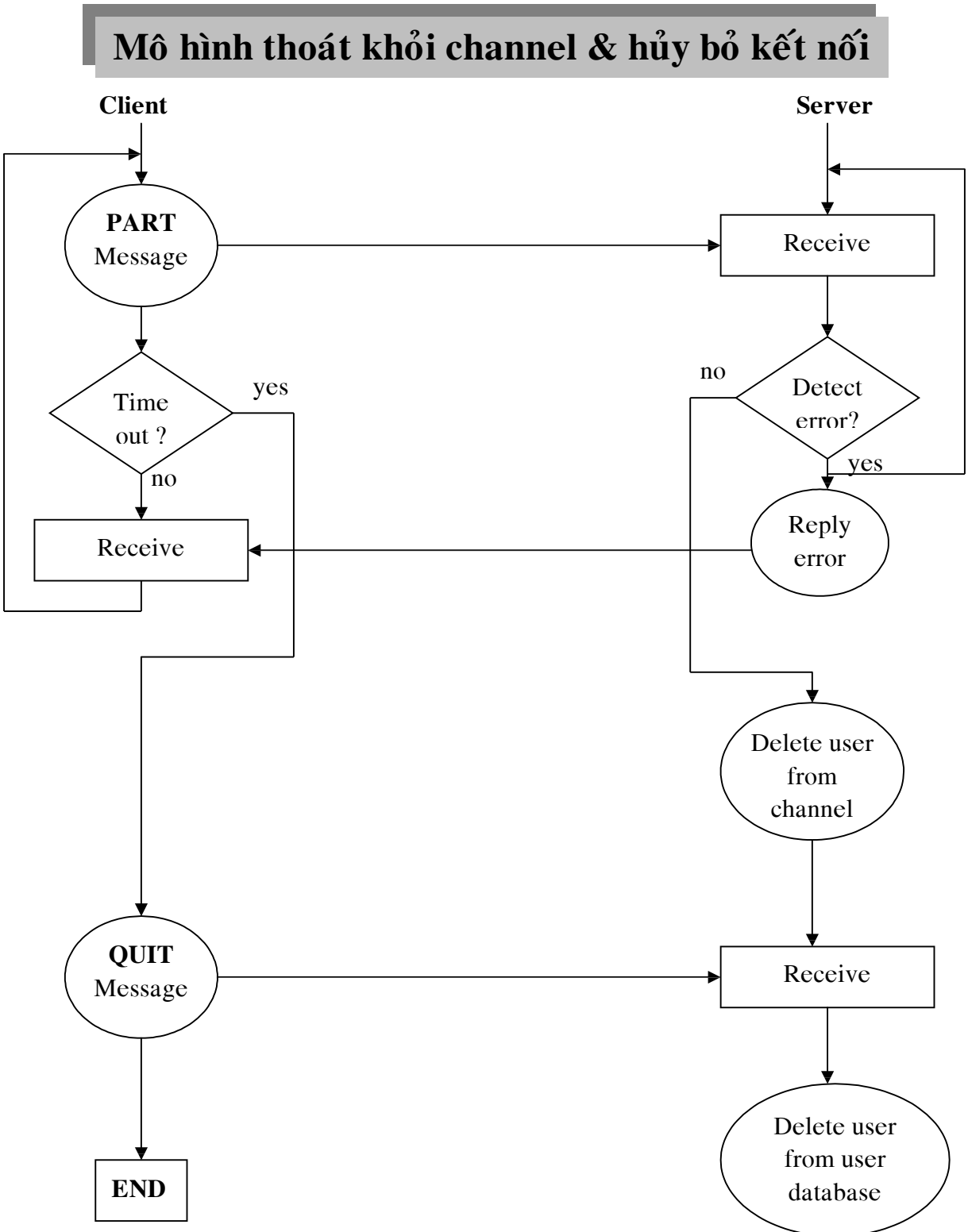
Hình 4: *Mô hình kết nối giữa chat client và chat server(còn tiếp)*



Hình 5: *Mô hình kết nối giữa chat client và chat server*



Hình 6: *Mô hình hoạt động sau khi kết nối*



Hình 7: *Mô hình thoát khỏi channel & hủy bỏ kết nối*

Trong nhóm message này có 3 loại message:

- ❖ Message do server sử dụng
- ❖ Message do client sử dụng
- ❖ Message dùng chung

Message do Server sử dụng

1.1 Server Message

Cú Pháp: SERVER <servername><hopcount><info>

Server message dùng để thông báo cho các server khác trên toàn hệ thống biết, có thêm một server kết nối vào hệ thống mạng (IRCnetwork). Những thông tin về server này được chuyển cho các server khác trên mạng chứa trong đối số <info>. Sự kiện này xảy ra khi có một server mới kết nối vào hệ thống mạng. Lúc đó server (mới kết nối vào) phát ra SERVER message, message này được phát quảng bá (broadcast) lên mạng. Hopcount thông tin về lượng giá Xem H4 để biết thêm chi tiết.

Lưu ý SERVER message khi gởi đến phải được chấp nhận bởi Server có tên là đối số <servername>.

Giải thích: vì chúng ta biết rằng khi có một server tham gia vào hệ thống thì các server khác phải có thông tin về server đó để chúng cập nhật lại cấu hình mạng, như thế thông tin này phải thông báo cho các server khác. Message này được gởi cho từng server.

Lỗi có thể xảy ra khi nhận được SERVER message, khi xảy ra lỗi server dùng ERROR message thông báo thay vì dùng giá trị trả về.

Giá trị trả về:

ERR_ALREADYREGISTRED

Ví dụ:

✓ SERVER test.oulu.fi 1 :[tolsun.oulu.fi]

<servername> : “test.oulu.fi” //server này cố gắng định danh trên mạng (IRCnetwork).

<hopcount>: “1”

<info>: “[tolsun.oulu.fi]”

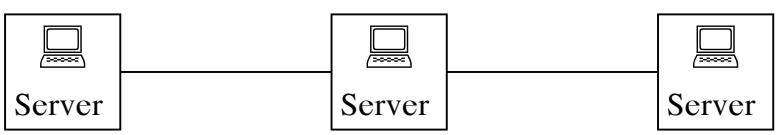
✓:tolsun.oulu.fi SERVER csd.bu.edu 5: BU Central Server

server: “tolsun.oulu.fi” không kết nối trực tiếp với server: “csd.bu.edu 5” khoảng cách là 5 hopcount.

1.2 Server Quit Message

Cú Pháp:

SQUIT <server><comment>



SQUIT message được dùng khi server muốn thoát khỏi hệ thống mạng. Nếu server B muốn hủy bỏ kết nối đến server A thì server B phải gởi SQUIT message đến Server A và Server C dùng tên serverB, server C làm đối số.

Message này cũng là hành động của operator giúp giữ cho hệ thống mạng gọn hơn (orderly fashion), hoặc loại bỏ server ra khỏi hệ thống nếu nó bị treo (deal). Operator có thể dùng SQUIT message cho các server ở xa trong tình trạng này các server còn lại phải phân tích SQUIT message này để cập nhật lại thông tin về hệ thống mạng(IRC Network). <comment> lí do việc tách khỏi hệ thống, comment được operator đưa ra.

Cả hai server A và server C phải phát ra SQUIT để thông báo đến tất cả server hiện vẫn còn kết nối với chúng, nhằm thông báo cho biết serverB không còn nữa, việc làm này giúp cho các server trên toàn mạng cập nhật lại cấu hình hiện tại.

Giá trị trả về: ERR_NOPRIVILEGES
ERR_NODUCHSERVER

Ví dụ:

✓ SQUIT tolsun.oulu.fi :Bad Link? Server đang kết nối với "tolsun.oulu.fi" phát ra thông báo.

✓ Trillian SQUIT cm22.eng.umd.edu : Server out of control

Message từ "Trillian" thông báo ngưng kết nối với <servername> "cm22.eng.umd.edu" vì lý do "Server out of control".

Message Client sử dụng

1.3 Operator message

Cú Pháp: OPRE <user><password>

Người dùng thông thường sử dụng dòng lệnh này để tranh quyền làm operator. Tuy nhiên, nếu server không cấu hình cho phép kết nối từ một *client* được phép tự thiết lập quyền làm operator khi gia nhập channel thì dòng lệnh này không có tác dụng (khi đó server sẽ trả về lỗi ERR_NOOPERHOST cho user).

Khi lệnh OPER thành công, server sẽ trả về cho user đó một thông điệp như khi sử dụng lệnh MODE đặt trạng thái operator cho người dùng.

Giá trị trả về:
ERR_NEEDMOREPARAMS
ERR_NOOPERHOST
RPL_YOUROPER
ERR_PASSWDMISMATCH

Ví dụ : ✓ OPRE foo bar
Username: "foo"
Password: "bar"

1.4 Quit message

Cú Pháp: QUIT [<Quit message>]

Một client muốn thoát ra hệ thống mạng nó phát ra QUIT message đến server mà nó kết nối trực tiếp server khi nhận được message này nó sẽ đóng kết nối với client đó. Đồng thời thông báo cho server khác để các server này thông báo cho các client trong cùng channel.

Giá trị trả về:

Không có

Ví dụ:

✓ QUIT :gone to have lunch

Nhóm message dùng chung

1.5 Pass message

Cú pháp : PASS < password>

Đối với Server:

Server dùng pass message để thiết lập password cho mình, bằng cách dùng password server ngăn không cho sự xâm nhập bất hợp pháp của các client khác. Server dùng pass message (có cung cấp password) trước khi server phát ra message SERVER(message này gúp cho server định danh trên mạng). Sau khi phát ra pass message server có thể đăng ký kết nối (connection register).

Đối với Client :

Client dùng pass message để thiết lập password cho nickname mà nó muốn đăng ký, nói cách khác. Password được dùng để không cho người khác sử dụng nickname của mình. Tất nhiên, việc thiết lập password là không cần thiết, nếu như chúng ta không cần bảo vệ nickname của mình. Tuy nhiên đối với nhóm channel operator là cần thiết. Cũng giống như server, client phải đưa ra PASS message trước khi đăng ký kết nối (bằng cách gởi NICK/USER message).

Lưu ý : PASS message có thể được gởi nhiều lần. Nhưng chỉ có lần cuối cùng gởi mới được xác nhận password và xem đó là password chính thức.

Giá trị trả về:

ERR_NEEDMOREPARAMS
ERR_ALREADYREGISTRED

Ví dụ:

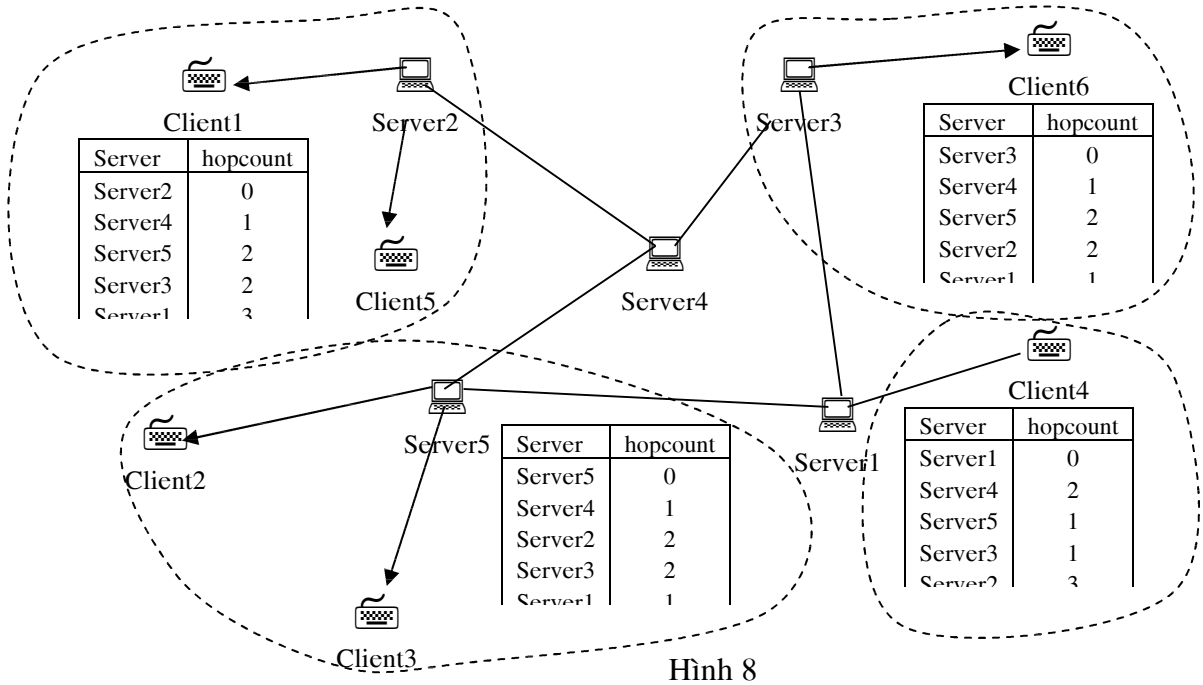
✓ PASS abc Thiết lập pass word “abc”

1.6 NickMessage

Cú pháp NICK <nickname>[<hopcount>]

Đối với Server:

Server dùng NICK message để thông báo vị trí của <nickname> cho các server khác, khi sever dùng NICK message đối số <hopcount> phải được đưa vào, (hop count là số server phải đi qua để đến được đích). Nếu một package được chuyển trên mạng khi đi qua một server nó trừ đi hopcount một đơn vị cho đến khi hopcount bằng 0 thì đến đích. Để hiểu rõ hơn về hop count xem hình bên dưới.



Hình 8

Đối với Client:

Client dùng NICK message để xác định nickname cho mình hoặc thay đổi nickname của mình. Nếu NICK message dùng cho client không cần đối số <hopcount>. Trong trường hợp có xuất hiện đối số <hopcount> thì nó sẽ bị bỏ qua.

Nếu NICK message từ client đến server, mà server đó đã có thông tin về nickname (nickname đã đăng ký) đó, thì hiện tượng cạnh tranh (Nickname Collision) xảy ra. Vì chúng ta được biết nickname phải là tên duy nhất không cho phép trùng. Kết quả là tất cả thông tin về nickname đó sẽ bị hủy bỏ khỏi cơ sở dữ liệu trong server(client kết nối) và một KILL message được phát ra để hủy bỏ nickname đó ra khỏi cơ sỡ dữ liệu của các server còn lại. Nếu server nhận được nickname từ client (kết nối trực tiếp vào server) mà nickname này bị trùng với một nickname hiện có trong server, thì nó sẽ phát ra thông báo lỗi ERR_NICKCOLLISION. Sau đó hủy bỏ NICK message và không phát ra KILL message. Như thế một client khi đưa ra nickname không thích hợp thì client đó không thể đăng ký kết nối với server đồng thời không ảnh hưởng đến các client khác.

Giá trị trả về: ERR_NONICKNAMEGIVEN
ERR_NICKNAMEINUSE
ERR_ERRONEUSNICKNAME

ERR_NICKCOLLISION

Vídụ: ✓ NICK Wiz : tạo ra một nickname mới tên là Wiz.
✓ Wiz NICK Kilroy :Thay đổi tên Wiz thành Kilroy.

1.7 User Message

Cú pháp:

USER <username><hostname><servername><realname>

Đối với Server:

Server dùng USER message để thông báo cho các server khác có user mới tham gia vào mạng (IRC network). Đồng thời nó cũng cung cấp cho các thông tin như <username>, <hostname>, <servername>, <realname>. Khi gởi USER message cho các server khác thì client sẽ gắn nickname vào trước message đó. Căn cứ vào nickname này, server sẽ nhận biết được thông tin thuộc về nickname nào. Nhưng có một điều lưu ý là server dùng USER message, sau khi nó nhận được NICK message và USER message từ client muốn kết nối vào hệ thống mạng. Điều này có nghĩa là client gởi NICK message thành công, tiếp đó là USER message được gởi đến server. Khi đó server sẽ phát ra USER message.

Đối với Client:

Sau khi đăng ký nickname thành công, client phát ra USER message để cung cấp thông tin cho server, thông tin mà nó cung cấp là <username>, <hostname>, <servername>, <realname>.

Tuy nhiên có một số điều lưu ý là: hai đối số <hostname>và <servername> bị bỏ đi nếu là client kết nối trực tiếp đến server. Vì lý do bảo mật trên mạng (security reasons). Và điều lưu ý thứ hai là đối số <realname> phải đứng sau cùng. Vì trong <realname> có thể có ký tự khoảng trắng và ký tự đầu tiên là dấu ':'.

Giá trị trả về: ERR_NEEDMOREPARAMS
ERR_ALREADYREGISTRED

Vídụ:

✓ USER guest tolmoon tolsun :Ronnie Reagan.
<Username>: "guest".
<Hostname>: "tolmoon".
<Servername>: "tolsun".
<Realname> :":Ronnie Reagan". -trong real name chúng ta nhận thấy có ký tự khoảng trắng và ký tự ':' đứng trước.

2. Nhóm message dùng cho việc điều khiển Channel

2.1 Join message

Cú Pháp:

JOIN <channel>{,<channel>} [<key>][,<key>]

Để có thể tham gia vào channel client phải phát ra JOIN message, server kiểm tra message này nếu cung cấp đúng channel thì cho phép gia nhập vào channel. Ngoài ra server còn

kiểm tra client đó có bị trạng thái "active bans", (đây là trạng thái mà client đó bị cấm quyền kết nối người cấm quyền chính là channel opertor) nếu gặp trạng thái này server không cho gia nhập.

Điều kiện để một client có thể tham gia vào channel:

- User phải được mời(invite) nếu channel đó đang ở trạng thái invite-only(mode +i). Nickname, username, hostname của user phải không ở vào trạng thái "active bans"(trạng thái cấm tham gia).
- Phải cung cấp đúng password và nickname nếu có thiết lập password cho nickname.
- Kể từ khi client JOIN thành công chúng sẽ nhận được thông báo, và có thể dùng các command để liên lạc với server của chúng bao gồm:MODE, KICK, PART, QUIT và quan trọng nhất là PRIVMSG/NOTICE message.
- JOIN message cũng cần phải được phát quảng bá(broadcast) lên mạng để các server có được thông tin về client mới tham gia, nhờ vào thông tin này các server có thể tìm thấy client đó.

Giá trị trả về : ERR_NEEDMOREPARAMS
ERR_INVITEONLYCHAN
ERR_CHANNELLISFILL
ERR_NOSUCHCHANNEL
RPL_TOPIC
ERR_BANNEDFROMCHAN
ERR_BADCHANNELKEY
ERR_BADCHANMASK
ERR_TOOMANYCHANNELS

Vídụ:

✓ JOIN #foobar Tham gia vào channel "#foobar".

✓ JOIN #foo fubar Tham gia vào channel "#foo" dùng khóa "fubar".

✓ JOIN #foo, &bar fubar Tham gia vào channel "#foo" dùng khóa "fubar" và "&bar"không dùng khóa.

✓ JOIN #foo, #bar, fubar, foobar Tham gia vào channel "#foo" , "#bar" dùng khóa "fubar", "foobar".

✓ :Wiz JOIN #Twilight_zone JOIN message từ WIZ.

2.2 part message

Cú pháp:

PART <channel>{,<channel>}

Client dùng part message để thoát khỏi channel

Giá trị trả về :

ERR_NEEDMOREPARAMS
ERR_NOTONCHANNEL

ERR_NOSUCHCHANNEL

Vídụ:

- ✓ PART #twilight_zone user thoát ra channel “#twilight_zone”.
- ✓ PART #oz-ops,&group5 rời bỏ cả 2 channel “#oz-ops”, “&group5”.

2.3 Mode message

Cú pháp chung: MODE

MODE message phục vụ 2 mục đích trong IRC, nó cho phép user và channel thay đổi mode của mình.

2.3.1 Channel mode

Cú pháp:

MODE <channel>{[+ | -]| o | p | s | i | t | n | b | v } [<limit>][<user>][<banmask>]

Trong trường hợp này MODE dùng thay đổi trạng thái channel chỉ có nhóm channel operator mới có quyền sử dụng command này.

Các giá trị mode:

o	-lấy quyền channel operation.
p	-cờ private.
s	-cờ secret.
i	-cờ thông báo channel ở trạng thái invite.
b	-thiết lập user ban mask.

Ví dụ:

- ✓ MODE #Finnish +im ; Makes #Finnish channel moderated and 'invite-only'.
- ✓ MODE #Finnish +o Kilroy ; Gives 'chanop' privileges to Kilroy on channel #Finnish.
- ✓ MODE #Finnish +v Wiz ; Allow WiZ to speak on #Finnish.
- ✓ MODE #Fins -s ; Removes 'secret' flag from channe #Fins.
- ✓ MODE #42 +k oulu ; Set the channel key to "oulu".
- ✓ MODE #eu-opers +l 10 ; Set the limit for the number of users on channel to 10.
- ✓ MODE &oulu +b ; list ban masks set for channel.
- ✓ MODE &oulu +b *!*@* ; prevent all users from joining.

✓ MODE &oulu +b *!*@*.edu ; prevent any user from a hostname matching *.edu from joining.

2.3.2 User mode

Cú pháp :

MODE <nickname>{[+ | -]| i | w | s | o |}

MODE trong trường hợp này dùng cho user.

Một user MODE chỉ có thể được chấp nhận nếu nickname người gởi và đối số <nickname> phải giống nhau.

Các giá trị mode:

i - cờ cho biết user đang ở mode invisible

s -

w –

o –

Lưu ý: một channel operator có thể tự giáng cấp (deopping) của mình bằng cách đưa ra mode -o

Giá trị trả về:

ERR_NEEDMOREPARAMS
ERR_CHANOPRIVSNEEDED
ERR_NOTONCHANNEL
RPL_BANLIST
ERR_UNKNOWNFLAG
ERR_USERSDONTMATCH
ERR_UMODEUNKNOWNFLAG
RPL_CHANNELMODEIS
ERR_NOSUCHNICK
ERR_KEYSET
RPL_ENDOFBANLIST
ERR_NOSUCHCHANNEL
RPL_UMODEIS

Ví dụ:

✓ :MODE WiZ -w ; turns reception of WALLOPS messages off for WiZ.

✓ :Angel MODE Angel +i ; Message from Angel to make themselves invisible.

2.4 Topic message

Cú Pháp:

TOPIC <channel>[<topic>]

Topic message dùng thay đổi hoặc xem chủ đề của channel. Nếu đối số <channel> được đưa vào lệnh TOPIC message sẽ thay đổi topic cho channel đó, trong trường hợp nó đang ở chế độ cho phép user thay đổi.

Giá trị trả về :

ERR_NEEDMOREPARAMS
RPL_NOTOPIC
ERR_CHANOPRIVSNEEDED
ERR_NOTONCHANNEL
RPL_TOPIC

Ví dụ:

- ✓ :Wiz TOPIC #test :New topic user Wiz setting topic.
- ✓ TOPIC #test : another topic chủ đề trên channel #test bằng "another topic".
- ✓ TOPIC #test kiểm tra(xem) topic trên channel "test".

2.5 Names message

Cú pháp:

NAMES [<channel>]{,<channel>}

Bằng cách sử dụng NAMES message. User có thể xem danh sách tất cả nickname có thể thấy được (visible) trên channel, đối số <channel> được dùng đối với trường hợp này không phải là channel private (mode+p) hoặc secret (mode +s), vì hai mode này không cho phép user truy xuất. Nếu có cung cấp đối số <channel> thì nó sẽ trả về danh sách nickname của channel đó, không có thông báo lỗi được trả về nếu sai channelname. Nếu trường hợp không có đối số <channel> thì nó sẽ trả về danh sách tất cả các channel và nickname (trong các channel đó).

Giá trị trả về:

RPL_NAMREPLY
RPL_ENDOFNAMES

Ví dụ:

- ✓ NAMES #twilight_zone ,#42 liệt kê danh sách của những user trên channel #twilight_zone và channel #42.
- ✓ NAMES liệt kê tất cả các user và channel (có thể nhìn thấy).

2.6 List message

Cú Pháp:

LIST [<channel>{,<channel>}[<server>]]

LIST message dùng để client liệt kê danh sách các channel và topic của chúng. Nếu đối số <channel> không được cung cấp thì nó sẽ liệt kê các channel mà client đang tham gia. Đối với Private và secret channel chúng vẫn được liệt kê nhưng không hiển thị chủ đề.

Giá trị trả về :

ERR_NOSUCHSERVER
RPL_LIST
RPL_LISTSTART

RPL_LISTEND

Ví dụ:

- ✓ LIST Liệt kê tất cả các channel.
- ✓ LIST #twilight_zone,#42 liệt kê danh sách topic của channel "#twilight_zone", "#42".

2.7 Invite message

Cú Pháp:

INVITE <nickname><channel>

INVITE message được dùng để mời/gọi user tham gia vào channel, đối số <nickname> là nickname của user được mời/gọi. Tuy nhiên không có sự đòi hỏi user phải tham gia vào channel. User được mời phải nằm trong channel ở chế độ +i (invite-only).

Giá trị trả về:

ERR_NEEDMOREPARAMS
E RR_NOTONCHANNEL
ERR_CHANOPRIVSNEEDED
RPL_INVITING
ERR_NOSUCHNICK
ERR_USERONCHANNEL
RPL_AWAY

Ví dụ:

:Angle INVITE Wiz #Dust user "Angle" mời/gọi "Wiz" tham gia channel "#Dust".

INVITE Wiz #Twilight_zone mời/gọi "Wiz" tham gia vào "#Twilight_zone".

2.8 Kick message

Cú pháp:

KICK <channel><user>[<comment>]

KICK command dùng để loại bỏ user ra khỏi channel. Chỉ có channel operator mới có quyền dùng lệnh KICK, mỗi server nhận được KICK message nó kiểm tra thật chính xác trước khi nó hủy/loại bỏ user đó ra khỏi channel. Sau khi bị KICK user đó có thể tái kết nối, điều này khác với BANs nếu user bị BANs nó sẽ không được kết nối trở lại cho đến khi người quản trị channel cho phép.

Giá trị trả về:

ERR_NEEDMOREPARAMS
ERR_BADCHANMASK
ERR_NOTONCHANNEL
ERR_NOSUCHCHANNEL
ERR_CHANOPRIVSNEEDED

Ví dụ:

✓ KICK &Melbourne Mattew : loại bỏ "Mattew" ra khỏi channel "&Melbourne".

✓ KICK #Finnish John :Speaking English "Jonh" sẽ bị loại tra khỏi channel "#Finnish" lí do "Speaking English".

✓ :Wiz KICK #Finnish John : message từ "Wiz" yêu cầu loại bỏ "Jonh" ra khỏi channel "#Finnish".

Ghi chư :không thể mở rộng các đối số như sau:

<channel>{,<channel>}<user>{,<user>}[<comment>]

3. Nhóm message truy vấn đến server (server query and command)

Server Queries and command là nhóm message được thiết kế để trả về thông tin của tất cả những server mà chúng đang tham gia trong mạng (IRC network), server (được truy vấn) phải trả lời chính xác các truy vấn. Nếu trường hợp server có những thông tin trả về không chính xác lập tức server đó sẽ bị loại ra khỏi mạng cho đến khi nó được phục hồi.

3.1 Version message

Cú pháp:

VERSION [<server>]

Message này dùng để xem version chương trình của server, đối số <server> là server name của server từ xa(remote server) mà client không trực tiếp kết nối đến, nếu không có đối số <server> mặt định là server hiện hành.

Giá trị trả về:

ERR_NOSUCHSERVER

RPL_VERSION

Ví dụ:

✓:Wiz VERSION *.se message từ "Wiz" kiểm tra version server "* .se".

✓VERSION tolsun.oulu.fi Kiểm tra version server "tolsun.oulu.fi".

3.2 Stats Message

Cú pháp:

STATS [<query>[<server>]]

client dùng STATS message để truy vấn(query) thông tin từ server, thông tin nhận được có thể là con số thống kê nào đó.

Hoạt động của lệnh này mang tính độc lập cao mặc dù server phải trả về thông tin truy vấn (query). Khi có STATS message đến server, nó sẽ kiểm tra xem server đích (destination server), sau đó nó chuyển message cho server kế tiếp cho đến khi đến đích.

Chúng ta tìm hiểu thêm về truy vấn(query) một truy vấn(query) trong trường hợp này là một ký tự duy nhất. Sau đây là các giá trị truy vấn (query) :

☞ c – return a list of server which the server may connect to or allow connection from; (trả về danh sách server có khả năng kết nối)

☞ h - return a list of server which are either forced to be treated as leaves or allowed to act as hubs.

☞ i - returns a list of hosts which the server allows a client to connect from; (trả về danh sách các host mà có thể kết nối đến server)

☞ k - returns a list of banned username/hostname combinations for that server; (trả về danh sách user ở trạng thái ban)

☞ l - returns a list of the server's connections, showing how long each connection has been established and the traffic over that connection in bytes and messages for each direction; (trả về danh sách server và thời gian mỗi kết nối được thiết lập)

☞ m - returns a list of commands supported by the server and the usage count for each if the usage count is non zero; (trả về những command được hỗ trợ bởi server)

☞ o - returns a list of hosts from which normal clients may become operators; (trả về danh sách các user trở thành operator)

☞ y - show Y (Class) lines from server's configuration file; (chỉ ra cấu hình file)

☞ u - returns a string showing how long the server has been up. (trả về chuỗi cho biết bao lâu server được khởi động)

Giá trị trả về:

ERR_NOSUCHSERVER

RPL_STATSCLINE	RPL_STATSNLINE
RPL_STATSILINE	RPL_STATSKLINE
RPL_STATSQLINE	RPL_STATSLLINE
RPL_STATSLINKINFO	RPL_STATSUPTIME
RPL_STATSCOMMANDS	RPL_STATSOLINE
RPL_STATSHLINE	RPL_ENDOFSTATS

Ví dụ:

✓ STATS m Yêu cầu server hiện hành trả về danh sách command được hổ trợ bởi server.

✓ :Wiz STATS c eff.org "Wiz" yêu cầu server, "eff.org" trả vể danh sách các server có kết nối với nó.

3.3 Link Message

Cú Pháp:

LINKS [[<remote server>]<server mask>]

Với message LINKS, user có thể liệt kê tất cả những server mà <remoteserver> biết. Danh sách trả về là các server. Tuy nhiên danh sách này có một số phần bị che dấu đi (mask) nếu không có sự che dấu này thì tất cả được hiển thị.

Giá trị trả về :

ERR_NOSUCHSERVER

RPL_LINKS

RPL_ENDOFLINKS

Ví dụ:

✓ LINKS *.au Liệt kê tất cả sác server có tên "*.au"

✓ :Wiz LINKS *.bu.edu *.edu message từ "Wiz " yêu cầu server "*.bu.ed" liệt kê tất cả các server có tên "*.edu".

3.4 Time Message

Cú pháp:

TIME [<server>]

TIME message dùng cho client truy vấn(query) về thời gian từ một server nào đó, server được truy vấn sẽ nằm trong đối số <server>, trường hợp đối số <server> không được đưa vào, thì server hiện hành phải trả lời truy vấn đó.

Giá trị trả về:

ERR_NOSUCHSERVER
RPL_TIME

Ví dụ:

✓ TIME tolsun.oulu.fi Truy vấn thời gian trên server "tolsun.oulu.fi".

✓ :Angle TIME *.au user "Angle" truy vấn thời gian từ server "*.au".

3.5 Connect message

Cú pháp:

CONNECT <target server>[<port>[<remote server>]]

CONNECT command có thể được dùng để buột server thiết lập một kết nối đến server khác ngay lập tức. CONNECT là lệnh hạn chế nó chỉ được dùng cho người quản trị ngoài ra không có client nào khác sử dụng được.

Giá trị trả về:

ERR_NOSUCHSERVER
ERR_NEEDMOREPARAMS
ERR_NOPRIVILEGES

Ví dụ:

✓ CONNECT tolsun.oulu.fi cố gắng kết nối với server "tolsun.oulu.fi".

✓ :Wiz CONNECT eff.org 6667 csd.bu.edu Wiz yêu cầu server "eff.org" và " csd.bu.edu " kết nối trên port 6667.

3.6 Trace Message

Cú pháp:

TRACE [<server>]

Đây là lệnh được dùng để tìm đường đi đến server nào đó (destination server). Mỗi server khi xử lý TRACE, message nó phải báo cho người gởi (sender) bằng cách là đưa ra những message chỉ đường, đồng thời nó gởi tiếp cho server kế tiếp. Tiến trình này được lặp đi lặp lại cho đến đích.

Nếu đối số <server> không đưa vào thì mặc định message sẽ đến server kết nối trực tiếp.

Lưu ý: trong khoảng giữa server gởi và server nhận, các server trung gian phải gởi trả về message trả lời RPL_TRACELINK.

Giá trị trả về:

RPL_TRACELINK
RPL_TRACKCONNECTING
RPL_TRACKUNKNOWN

RPL_TRACKUSER
RPL_TRACKSERVICE
RPL_TRACKCLASS
RPL_TRACKHANDSHAKE
RPL_TRACKOPERATOR
RPL_TRACKSERVER
RPL_TRACKNEWTYPE

Ví dụ:

- ✓ TRACK *.oulu.fi — TRACK đến server "*.oulu.fi".
- ✓ :Wiz TRACK AngelDust — Track được đưa ra bởi "Wiz" đến "AngelDust".

3.7 Admin Message

Cú pháp:

ADMIN [<server>]

Server dùng ADMIN message để tìm tên người quản lý (administrator) của <server> dùng làm đối số.

Nếu không có đối số <server> thì xemserver mà nó kết nối trực tiếp đến là đối số. Mỗi server có khả năng chuyển ADMIN message đến server khác, sao cho message đến được server cần đến.

Như thế chúng ta thấy rằng một client có thể yêu cầu một server khác gởi thông tin về người quản lý cho mình

Giá trị trả về:

ERR_NOSUCHSERVER
RPL_ADMINME
RPL_ADMINLOC1
RPL_ADMINLOC2
RPL_ADMINEMAIL

Ví dụ:

- ✓ ADMIN tolsun.oulu.fi — yêu cầu server "tolsun.oulu.fi" trả về admin
- ✓ :Wiz ADMIN *.edu — Wiz yêu cầu

3.8 Info Message

Cú pháp:

INFO [<server>]

INFO message do client yêu cầu <server> khác trả về thông tin cho mình, nếu đối số <server> không đưa ra xem như server hiện tại(server kết nối trực tiếp).

Giá trị trả về:

ERR_NOSUCHSERVER
RPL_ENDOFINFO
RPL_INFO

Ví dụ:

- ✓ INFO csd.bu.edu — yêu cầu thông tin trả về từ "csd.bu.edu".

✓ :Avalon INFO *.fi "Avalon" yêu cầu thông tin trả về từ "*.fi".

✓ INFO Angle yêu cầu server mà user "Angle" kết nối đến, trả về thông tin.

4. Nhóm message gởi text(sending message)

Mục đích chính IRC protocol là cung cấp cho các client có thể giao tiếp với nhau, để thực hiện việc trao đổi text, ta cần có Private Message và Notice Message để chuyển text message từ client này đến client khác. Sau khi thực hiện quá trình kết nối với server và gia nhập channel thành công sending message sẽ hoạt động để thực hiện mục đích chính cho chat protocol.

4.1 Private Message

Cú pháp:

PRIVMSG <receiver>{,<receiver>}<text to be sent>

Message này dùng cho client, user có thể gởi private message từ user gởi đến user nhận, và chỉ có user nhận mới được nhận PRIVMSG message. Đối số <receiver> là nickname của client nhận, <receiver> có thể là danh sách nickname, channel chúng được phân biệt với nhau bằng dấu phẩy (' , ').

Giá trị trả về:

ERR_NORECIPIENT
ERR_CANNOTSENDTOCHAN
ERR_WILLTOPLEVEL
ERR_NOTOPLEVEL
ERR_NOSUCHNICK
ERR_NOTEXTTOSEND
ERR_TOOMANYTARGETS
RPL_AWAY

Vídụ:

✓ :Angle PRIVMSG Wiz :Hello are you receiving this message?
Message từ "Angle" đến "Wiz".

✓ PRIVMSG Angle :Yes I'm receiving it !receiving it !'u>(768u+1n).br
Message đến Angle.

✓ PRIVMSG jto@tolsum.oulu.fi: Hello! Message từ client trên server" tolsum.oulu.fi" vớo user name "jto".

✓ PRIVMSG $*.fi :Server tolsun.oulu.fi rebooting Message đến tất cả user trên server có tên "*.fi".

✓ PRIVMSG #*.edu :NFSNet is undergoing work, expect interruptions
Message đến tất cả user có tên "*.edu".

4.2Notice Message

Cú pháp:

NOTICE <nickname><text>

NOTICE message sử dụng tương tự như PRIVSMSG message chỉ có điều khác nhau giữa hai message này là NOTICE message không cần phải có sự đáp lại từ client <nickname> nhận. Qui định này cũng được áp dụng cho server, như thế server không cần thông báo lỗi cho client. Lý do có việc quy định này là để tránh sự tự động trả lời lặp đi lặp lại giữa server và client.

Giá trị trả về:

ERR_NORECIPIENT
ERR_CANNOTSENDTOCHAN
ERR_WILLTOPLEVEL
ERR_NOTOPLEVEL
ERR_NOSUCHNICK
ERR_NOTEXTTOSEND
ERR_TOOMANYTARGETS
RPL_AWAY

Vídụ:

✓ :Angle NOTICE Wiz :Hello are you receiving this message? Message từ "Angle" đến "Wiz".

✓ NOTICE Angle :Yes I'm receiving it !receiving it !'u>(768u+1n).br Message đến Angle.

✓ NOTICE jto@tolsum.oulu.fi : Hello! Message từ client trên server" tolsum.oulu.fi" với user name "jto".

✓ NOTICE $*.fi :Server tolsun.oulu.fi rebooting Message đến tất cả user trên server có tên "*.fi".

✓ NOTICE #*.edu :NFSNet is undergoing work , expect interruptions Message đến tất cả user có tên "*.edu".

5. Nhóm message do client truy vấn đến server(user-based query)

Đây là nhóm command mà user có thể dùng để tạo truy vấn. Khi dùng nhóm lệnh này user có thể xem được thông tin chi tiết về các user khác với điều kiện các user có thể nhận biết nhau trên mạng. Việc có thể nhận biết nhau tùy thuộc vào mode user đó và trạng thái channel đang dùng.

5.1 Who Query

Cú pháp:

WHO [<name>[<o>]]

WHO message dùng cho client để tạo ra truy vấn, kết quả truy vấn đó là danh sách các user phù hợp với đối số <name>. Nếu không có đối số <name> thì tất cả user (có thể thấy được) được trả về, có thể nhận được danh sách các operator nếu chúng ta sử dụng đối số [<o>]hoặc ký tự đại diện(wildcard).

Đối số <name> có thể là hostname, server, realname, nickname.

Giá trị trả về:

ERR_NOSUCHSERVER
RPL_ENDOFWHO
RPL_WHOREPLY

Ví dụ:

✓ WHO *.fi yêu cầu liệt kê tất cả user có tên "*.fi".

✓ WHO jto* o yêu cầu liệt kê tất cả user có tên "jto*" nếu chúng là operator.

5.2 Who is Query

Cú pháp:

WHOIS [<server>]<nickmask>[,<nickmask>[,...]]

Message này dùng để yêu cầu thông tin cụ thể về một user nào đó, như thế có sự khác nhau giữa WHO message và WHOIS message cụ thể là:

WHO message: yêu cầu danh sách các user.

WHOIS message: yêu cầu thông tin về một user nào đó.

Server sẽ trả lời cho message này, trong trường hợp server là server ở xa(remote server) thì nó phải được đưa vào đối số <server>.

Lưu ý :Danh sách các nickmask phải cách nhau bởi dấu phẩy.

Giá trị trả về:

ERR_NOSUCHSERVER
ERR_NONNICKNAMEGIVEN
RPL_WHOISUSER
RPL_WHOISCHANNELS
RPL_WHOISSERVER
RPL_AWAY
RPL_ WHOISOPERATOR
RPL_WHOISIDLE
RPL_ENDOFWHOIS
ERR_NOSUCHNICK

Ví dụ:

✓ WHOIS wiz trả về thông tin user hiện hành với nickmask " wiz".
✓ WHOIS eff.org trillian truy vấn đến server "eff.org" về "trillian".

5.3 Whowas Message

Cú pháp:

WHOWAS <nickname>[<count>[<server>]]

Khác với WHOIS messge, WHOWAS messge yêu cầu thông tin những user trong quá khứ mà hiện giờ không còn có mặt trong danh sách user active. Để trả lời cho message này server phải tìm kiếm trong danh sách các nickname đã thoát ra khỏi hệ thống mạng(IRC network). Chúng ta thấy trong danh sách đối số có <count>, đối số này là một số nguyên dương, cho biết lấy bao nhiêu lần thông tin về nickname đó, có nghĩa là trong danh sách các nickname đã rời khỏi hệ thống mạng có thể có nhiều mẩu tin nickname giống nhau nhưng khác nhau về thời gian đăng nhập. Khi đó <count> quy định lấy bao nhiêu mẩu tin, nếu trường hợp không có giá trị <count> server sẽ trả về toàn bộ thông tin các nickname mà server có được.

Giá trị trả về:

ERR_NONICKNAMEGIVEN
ERR_WASNOSUCHNICK
RPL_WHOWASUSER
RPL_WHOSISERVER

RPL_ENDOFWHOWAS

Ví dụ:

✓ WHOWAS wiz trả về tất cả những thông tin trong quá khứ về nickname "wiz".

✓ WHOWAS Mermaid 9 trả về thông tin trong quá khứ của "Mermaid" nhưng chỉ lấy tối đa 9 mẩu tin về nickname này.

✓ WHOWAS Trillian 1 *.edu chỉ lấy 1 mẩu tin về nickname "Trillian" ở server "*.edu".

6. Nhóm message khác (miscellaneous message)

6.1 Kill Message

Cú pháp:

KILL<nickname> <comment>

KILL message được dùng khi cần kết thúc kết nối giữa client và server, khi gặp phải hiện tượng Nick Collision(khi có sự trùng lắp của 2 mẩu tin trong danh sách Nickname.) xảy ra, khi đó server sẽ phát ra KILL message, KILL message cũng được channel operator sử dụng <comment> là lý do cho KILL message do server đưa ra để giải thích cho hành động của mình.

Giá Trị trả về:

ERR_NOPRIVILEGES
ERR_NEEDMOREPARAMS
ERR_NOSUCHNICK
ERR_CANTKILLSERVER

Ví dụ:

✓ KILL David(csd.bu.edu <- tolsun.olu.fi) giữa csd.bu.edu và tolsun.olu.fi xảy ra hiện tượng Nick Collition nên "David" bị ngưng kết nối.

Ghi chú: chỉ có operator mới có thể dùng KILL message để ngưng kết nối của users.

6.2 Ping Message

Cú Pháp:

PING <server1>[<server2>]

PING message dùng để kiểm tra một client còn hoạt động trong hệ thống mạng hay không? PING message được server gởi đến client, khi nhận được message này, client phải đáp lại <server> bằng PONG message, nếu sau một khoảng thời gian nào đó(time out) mà không thấy client trả lời thì xem như kết nối đó bị ngắt, vì thế khi nhận được PING message client phải trả lời bằng PONG message càng sớm càng tốt.

Xem trong danh sách đối số ta nhận thấy rằng có đến 2 đối số <server1> và <server2> điều đó có nghĩa là PING message gởi đến cả hai <server1> và <server2>, tuy nhiên server nhận được message nó không đáp lại mà dựa vào client kết nối với nó để thông báo kết nối vẫn còn

liên thông. Từ đây chúng ta rút ra được kết luận là: Server không đáp lại PING message mà nó nhờ client đáp lại.

Giá trị trả về:

ERR_NOORIGIN

ERR_NOSUCHSERVER

Ví dụ:

✓ PING tolsun.oulu.fi Server gởi PING message kiểm tra server khác còn hoạt động hay không.

✓ PING Wiz PING message được gởi đến Wiz.

6.3 Pong Message

Cú Pháp:

PONG <daemon>[<daemon2>]

PONG là message được client dùng để đáp lại PING message<daemon> là đối số cho biết client nào đáp lại PING message.

Giá trị trả về:

ERR_NOORIGIN

ERR_NOSUCHSERVER

Ví dụ:

✓ PONG csd.bu.edu tolsun.oulu.fi PONG message từ csd.bu.edu đến tolsun.oulu.fi.

6.4 Error Message

Cú pháp:

ERROR<errormessage>

ERROR message dùng cho server thông báo lỗi cho operator. Nó có thể được gởi từ server này đến server khác. Message này chỉ dùng để thông báo lỗi liên kết giữa server với nhau, nếu server nhận được ERROR message từ server khác thì nó không cần thông báo cho server khác biết, đều này có thể hiểu rằng server gởi ERROR message đến server nhận, khi đó message chỉ chuyển cho operator của nó.

Một khi cần thông báo lỗi đến client operator nó phải được đóng gói thông qua lệnh NOTICE message để chuyển, tất nhiên client operator không cần phải trả lời cho message này.

Giá trị trả về:

Không có

Ví dụ:

✓ ERROR :server *.fi already exists Thông báo lỗi chuyển đến server đến các khác

✓ NOTICE Wiz :ERROR from cds.bu.edu --Server *.fi already exists thông báo lỗi gởi đến Wiz trên server cds.bu.edu.

7. Nhóm message tùy chọn(option message)

7.1 Away Message

Cú pháp:

AWAY[message]

Với AWAY message user có thể thiết lập cơ chế trả lời tự động cho mỗi PRIVMSG message, nội dung trả lời được lưu trữ trong chuỗi [message]. Sau khi thiết lập xong chế độ này việc trả lời tự động sẽ được gởi từ server đến client phát ra PRIVMSG message, server gởi chính là server kết nối trực tiếp với client phát ra PRIVMSG message.

Nếu AWAY message không có [message] thì xem như chuỗi trả lời bị xóa bỏ.

Giá trị trả về:

RPL_UNAWAY
RPL_NOWAWAY

Ví dụ:

✓AWAY :Gone to have lunch. Back in 5 minute thiết lập việc trả lời tự động.
✓Wiz AWAY "Wiz" hủy bỏ chuỗi trả lời tự động.

7.2 Rehast Message

Cú pháp:

REHASH

Server có một tập tin cấu hình để thiết lập các tham số. Lệnh này do operator thi hành bắt buộc server phải đọc lại và xử lý lại tập tin cấu hình của server đó.

Giá trị trả về:

RPL_REHASHING
ERR_NOPRIVILEGES

Ví dụ:

✓REHAST message từ channel operator yêu cầu server đọc lại tập tin cấu hình.

7.3 Restart Message

Cú pháp:

RESTART

Đây là message dùng cho channel operator buộc server phải khởi động (restart) lại hệ thống.

Giá trị trả về:

ERR_NOPRIVILEGES

Ví dụ:

✓RESTART server hiện tại khởi động lại hệ thống.

7.4 Summon Message

Cú pháp:

SUMMON <user>[<server>]

Mời một client có chương trình server hiện đã có cài đặt trên host gia nhập hệ thống IRC

Nếu không có đối số <server> thì nó xem server hiện hành là server đích, khi đó các client kết nối trực tiếp đến server đó sẽ được mời.

Giá trị trả về :

ERR_NORECIPIENT
ERR_FILEERROR
ERR_NOLOGIN
ERR_NOSUCHSERVER
RPL_SUMMONING

Ví dụ:

- ✓ SUMMON jto — gọi user "jto" tham gia vào server Hiện hành.
- ✓ SUMMON jto tolsun.oulu.fi — gọi user "jto" tham gia vào server "tolsun.oulu.fi".

7.5 Users Message

Cú Pháp:

USER[<server>]

USER message dùng để trả về danh sách của các user đã login vào <server> tương tự như WHO message. Tuy nhiên có một số client không thể dùng message này được trên server của họ, có thể vì lý do bảo mật (security reasion). Nếu user không thể dùng message này một reply được phát ra.

Giá trị trả về:

ERR_NOSUCHSERVER
ERR_FILEERROR
RPL_USERSSTART
RPL_NOUSER
RPL_USERS
RPL_ENDOFUSERS
ERR_USERSDISABLED

Disabled reply:

ERR_USERDISABLED

Ví dụ:

- ✓ USER eff.org — yêu cầu danh sách các user đã login vào server "eff.org".
- ✓ :Jonh USER tolsun.oulu.fi "Jonh" yêu cầu danh sách user login trên server "tolsun.oulu.fi".

7.6 Operwall Message

Cú pháp:

WALLOPS<text to be sent to all operators currently online>

Khi WALLOPS hoạt động nó sẽ gởi <text .....> đến tất cả các operator hiện có trên mạng.

Giá trị trả về:

ERR_NEEDMOREPARAMS

Ví dụ:

- ✓ :csd.bu.edu WALLOPS :Connect '*.uiuc.edu 6667'from Joshua — Message từ "csd.bu.edu" thông báo CONNECT message đến các operator.

7.7 Userhost Message

Cú pháp:

USERHOST<nickname>{<space><nickname>}

USERHOST yêu cầu trả về thông tin của client có <nickname> làm đối số, danh sách <nickname> có thể lên đến 5 đối số khi đó thông tin về mỗi nickname sẽ được trả về cho client.

Lưu ý : mỗi thông tin trả về được phân cách bằng ký tự khoảng cách.

Giá trị trả về:

RPL_USERHOST

ERR_NEEDMOREPARAMS

Ví dụ:

✓ USERHOST Wiz Michael Marry P USERHOST yêu cầu thông tin về "Wiz","Michael","marty","P".

7.8 Ison Message

Cú pháp:

ISON<nickname>{<space><nickname>}

ISON message được thiết lập để trả về thông tin về <nickname> hiện thời trên IRC một cách hiệu quả nhất. Có thể có nhiều đối số <nickname> được đưa vào nhưng chúng phải cách nhau bằng ký tự khoản trắng.

Giá trị trả về:

RPL_ISON

ERR_NEEDMOREPARAMS

Ví dụ:

✓ ISON phone trillian Wiz Jarlek Avalon Angel Monstah ISON yêu cầu 7 host.

Phần II:
Tìm Hiểu Kỹ Thuật Lập Trình Socket

- ✓ Chương I: Các Khái Niệm Cơ Bản Về Hệ Thống Mạng
- ✓Chương II: Một Số Hàm Socket

Chương I: Các Khái Niệm Cơ Bản Về Hệ Thống Mạng

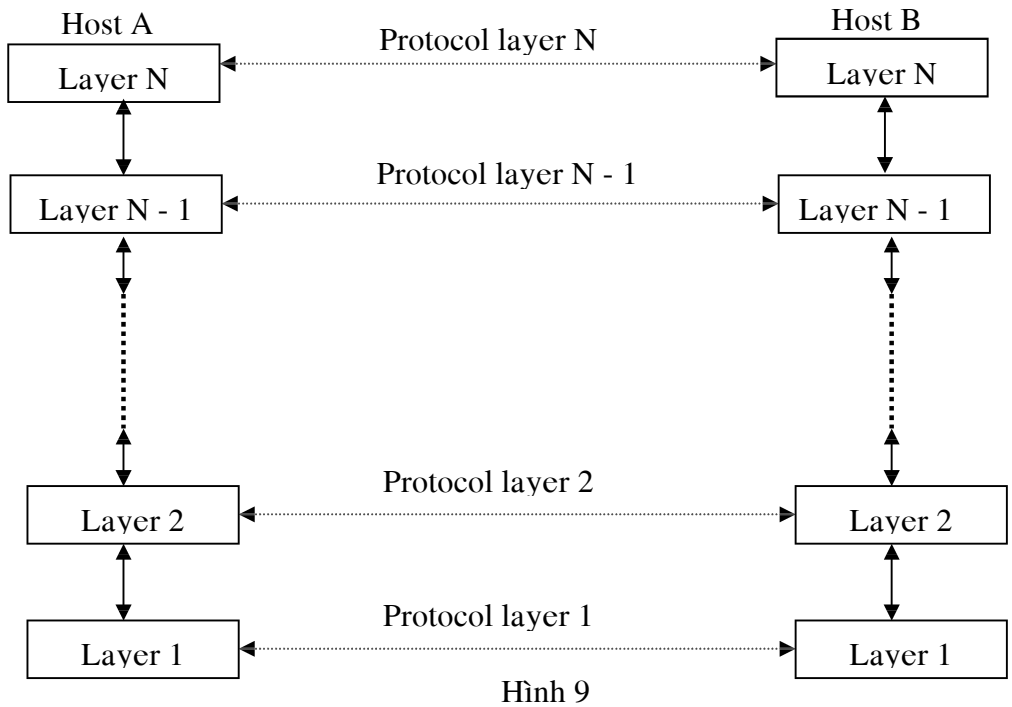
1. Mô hình mạng, mô hình OSI, mô hình TCP.

1.1 Mô Hình Mạng

Trước tiên chúng ta cần tìm hiểu vấn đề: tại sao có sự phân tầng của các protocol, ích lợi của việc phân tầng.

Để có thể chuyển một thông điệp (message) từ máy này sang máy khác (các máy phải trong cùng hệ thống mạng) nó phải trải qua nhiều giai đoạn khác nhau, các giai đoạn này rất phức tạp như là: chia nhỏ thông điệp (message) ra thành nhiều gói nhỏ (package), mã hóa các gói này ra dạng bit, các bit này được chuyển qua đường truyền vật lý đến máy nhận. Sau đó quá trình nhận sẽ thực hiện ngược lại như bên gởi, nếu quá trình lắp ghép gặp phải lỗi thì phải thông báo để truyền lại vv...

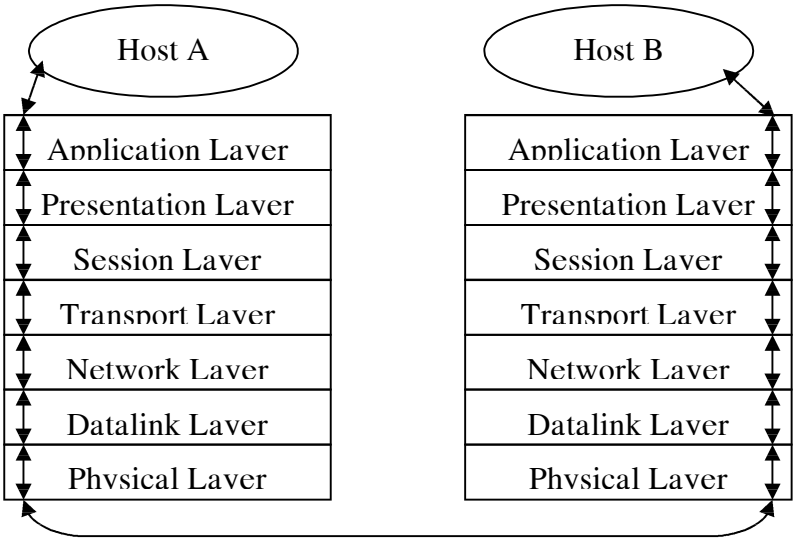
Các giai đoạn này rất phức tạp đòi hỏi người lập trình ứng dụng phải hiểu rõ tất cả các cơ chế hoạt động bên dưới của hệ thống. Vì vậy người ta đưa ra ý tưởng phân tầng, mỗi tầng sẽ chịu trách nhiệm cung cấp dịch vụ cho tầng bên trên và đồng thời nó cũng sử dụng dịch vụ của tầng bên dưới cung cấp cho nó. Như thế một người làm việc ở tầng nào họ chỉ quan tâm đến các tầng có quan hệ trực tiếp với mình.



Hình 9

Trong mô hình này mỗi lớp n + 1 sử dụng dịch vụ của lớp n, cả hai host A và host B phải có cùng chồng giao thức(protocol stack).

1.2 Mô hình OSI



Hình 10: *Mô hình OSI*

Ý nghĩa các tầng :

Physical Layer

Ở lớp này thông tin được truyền dưới dạng bit thông qua kênh truyền. Và nhận các bit chuyển lên cho lớp datalink.

Datalink Layer

Lớp này có nhiệm vụ chia nhỏ dữ liệu từ lớp network đưa xuống thành các frame, mỗi frame có dung lượng từ vài trăm byte đến vài ngàn byte. Các frame được truyền đi bằng cách chuyển xuống cho lớp physical. Nhiệm vụ thứ hai là tổ chức nhận các frame sao cho đúng thứ tự, cung cấp khả năng truyền không lỗi trên đường truyền vật lý cho các lớp cao hơn. Vấn đề đặt ra ở đây là phải xác định cơ chế để xác nhận một frame có truyền thành công hay không (Acknowladge Framje), xử lý nhiểu (truyền lại).

Network layer

Lớp này định hướng cho gói dữ liệu (package) đi từ máy gởi đến máy nhận. Phải giải quyết vấn đề định tuyến (routing), vấn đề địa chỉ (addressing), lượng giá chi phí (accouting), và giải quyết đụng độ (collision).

Transport layer

Lớp này có nhiệm vụ chia nhỏ gói dữ liệu được đưa xuống từ lớp bên trên thành những đơn vị nhỏ hơn để truyền qua mạng, với sự đảm bảo là dữ liệu đến nơi một cách chính xác. Lớp này cung cấp cho các lớp bên trên phương tiện để truyền các message độc lập với các lớp bên dưới.

Session player

Session layer điều khiển quá trình giao tiếp giữa hai tiến trình trên hai máy, tạo và kết thúc kết nối giữa các quá trình trên các máy khác nhau.

Presentation layer

Lớp này biểu diễn những thông tin được truyền (được hiểu là cú pháp và ngữ nghĩa), nó đồng nhất các thông tin giữa các các hệ thống khác nhau. Ngoài ra có còn cung cấp dịch vụ thao tác trên dữ liệu như nén, mã hóa(compression & cryptography).

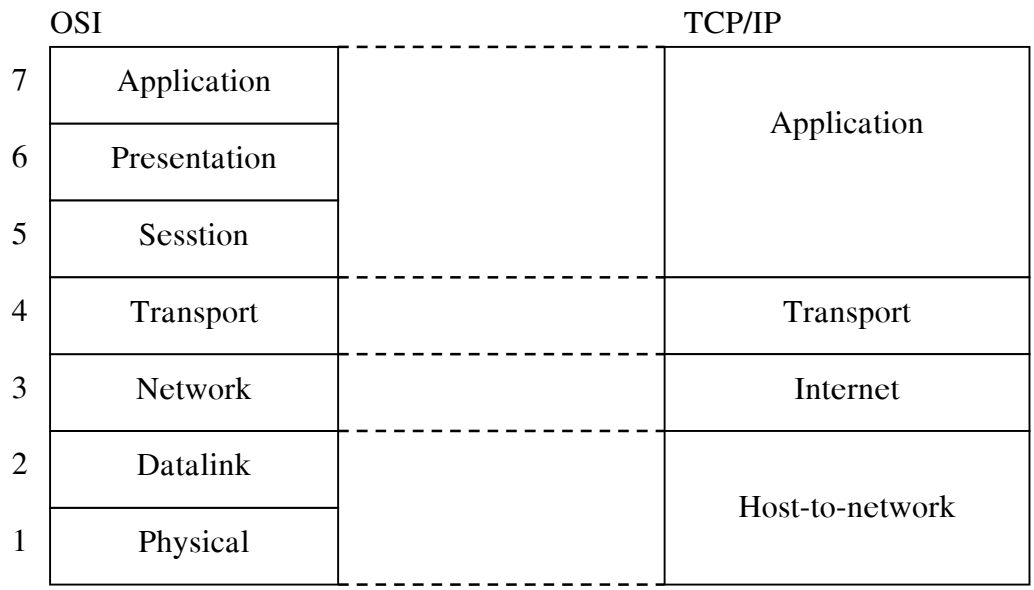
Application layer

Đây là lớp cung vấp dịch vụ cho người sử dụng (end user), ứng với mỗi dịch vụ (còn được gọi là ứng dụng) có 1 protocol khác nhau.

Ví dụ: FTP(truyền nhận file), HTTP, E-mai, v.v..

1.3 Mô hìnhTCP/IP

Chúng ta đã khảo sát mô hình OSI 7 lớp, mô hình này chỉ là mô hình tham khảo, việc áp dụng mô hình này vào thực tế là khó có thể thực hiện (hiệu suất kém vì dữ liệu khi truyền từ máy này sang máy kia trong mạng thì phải trải qua tất cả các lớp của mô hình OSI ở cả 2 máy), nó chỉ là tiêu chuẩn để các nhà phát triển dựa theo đó mà phát triển các mô hình khác tối ưu hơn. Có rất nhiều các mô hình khác nhau, hiện nay mô hình TCP/IP được sử dụng phổ biến nhất.



Hình 11: *Mô hình TCP/IP*

Bộ protocol TCP/IP bao gồm:

- TCP(Transmission Control Protocol): đây là loại protocol có cầu nối (connection oriented) cung cấp khả năng truyền dòng dữ liệu không lỗi, 2 chiều (full duplex)cho các quá trình cho người sử dụng.
- UDP(User Datagram Protocol): loại protocol không thiết lập cầu nối (connectionless) cho các quá trình của user. Không giống như TCP, nó không đảm bảo dữ liệu khi truyền đi có đến nơi chính xác hay không.

- ICMP (Internet Control Message Protocol): protocol xử lý lỗi và điều khiển thông tin giữa các gateway và các host.
- IP(Internet Protocol): IP là protocol cung cấp dịch vụ phân phối các package cho TCP, UDP và ICMP.
- ARP (Address Resolution Protocol): Protocol ánh xạ 1 địa chỉ internet thành địa chỉ phần cứng(MAC address).
- RAPR(Address Resolution Protocol): Protocol ánh xạ một địa chỉ phần cứng thành địa chỉ IP.

Mô hình TCP/IP được phân ra thành 4 lớp, trong đó 2 lớp dưới (1 và 2)của mô hình OSI được gộp lại thành 1 lớp gọi là Host-to-network; 2 lớp Session và Presentation của OSI không có trong mô hình giao thức TCP/IP.

Tương tự như mô hình OSI, trong mô hình TCP/IP, dữ liệu từ 1 máy cũng đi từ lớp Application xuống Transport, rồi xuống tiếp lớp Internet, sau cùng đi tới lớp Host-to-network, thông qua đường dây vật lý đến 1 máy khác trong mạng : dữ liệu ở đây sẽ đi ngược từ dưới lên. Cũng giống như mô hình OSI, ở đây, giữa các lớp của 2 máy giao tiếp với nhau thông qua một protocol; giữa lớp này với lớp khác trong cùng một máy gọi là Interface. Lớp bên dưới cung cấp các dịch vụ cho lớp trên.

Host-to-network

Kết nối host với network sao cho chúng có thể chuyển các message tới các địa chỉ đích, lớp này gần giống với lớp physical trong mô hình OSI.

Internet layer

Đây là lớp thực hiện một hệ thống mạng có khả năng chuyển mạch các gói dữ liệu dựa trên một lớp mạng Connectionless(không cầu nối) hay Connection – Oriented (có cầu nối) tùy vào loại dịch vụ mà người ta dùng một trong 2 cách trên.

Nhiệm vụ của lớp này là đảm bảo cho các host chuyển các package vào bất kì hệ thống mạng nào và chuyển chúng đến đích mà không phụ thuộc vào vị trí của đích đến.

Trong mô hình TCP/IP người ta đưa ra khái niệm địa chỉ IP để định địa chỉ cho các host trên mạng(xem phần địa chỉ IP).

Transport layer

Lớp transport được thiết kế để cho các phần tử ngang cấp ở lớp host có thể đối thoại với nhau.

Hai protocol chính là :

- TCP: là một Connection Oriented Protocol, cho phép chuyển một chuỗi byte từ host này sang host kia mà không có lỗi (dùng cơ chế phân chia dữ liệu ra thành các gói nhỏ(package) ở máy nguồn và gom lại ở máy đích).
- UDP: là một connectionless Protocol được xây dựng cho các ứng dụng không muốn sử dụng cách truyền theo một thứ tự của TCP mà muốn tự mình thực hiện điều đó (tùy theo mục đích của ứng dụng mà người ta dùng UDP hay không).

Khái niệm về port:

Một máy có thể liên lạc với một máy khác trong mạng qua địa chỉ IP. Tuy nhiên, với địa chỉ như vậy không đủ cho một process của máy liên lạc với một process của máy khác. Và vì vậy TCP/UDP đã dùng số nguyên (16 bit) để đặt tả nên số hiệu port.

Như vậy, để hai process của hai máy bất kỳ trong mạng có thể giao tiếp được với nhau thì mỗi frame ở cấp Network có IP gồm :

+ Protocol (TCP/UDP).
+ Địa chỉ IP của máy gởi.
+ Số hiệu port của máy gởi.
+ Địa chỉ IP của máy đích
+ Số hiệu port của process ở máy đích.

Ví dụ:{TCP,127.28.11.83,6000,127.28.11.241,7000};

Application layer(process layer)

Chứa các dịch vụ như trong các lớp Session, Presentaion, Application của mô hình OSI, ví dụ: Telnet(Terminel Acesss) cho phép user thâm nhập vào một host ở xa và làm việc ở đó như đang làm việc trên máy local(cục bộ), FPT (File Transfer Protocol) là công cụ giúp cho chúng ta chuyển các file cho nhau, SMTP(Simple Mail Transfer Protocol) cũng là một dạng của FTP nhưng nó đặc điểm riêng, DNS(Domain Name Service) dùng để ánh xạ tên host thành địa chỉ IP và ngược lại.

2. Giao Thức TCP và UDP

2.1 Giao Thức UDP

UDP là phương thức truyền dữ liệu theo phương pháp không hướng kết nối (connectionless). Khi truyền nó không cần thiết lập cầu nối giữa máy gởi và máy nhận, sử dụng cơ chế UDP người ta giả định rằng ở máy nhận luôn sẳn sàn đón nhận dữ liệu gởi đến. Nếu dữ liệu gởi đến bị lỗi trong quá trình truyền hay không nhận được đầy đủ, UDP cũng không có thông tin phản hồi lại cho máy gởi. Tuy nhiên UDP không đòi hòi nhiều tài nguyên của hệ thống và thiết kế chương trình ứng dụng đơn giản. UDP thường được dùng trong những ứng dụng không đòi hỏi độ chính xác cao ví dụ: dịch vụ thông báo giờ, tỉ giá, hay dịch vụ nhắn tin và dùng cho việc truyền tải những file có kích thước lớn như hình ảnh, âm thanh, vv.

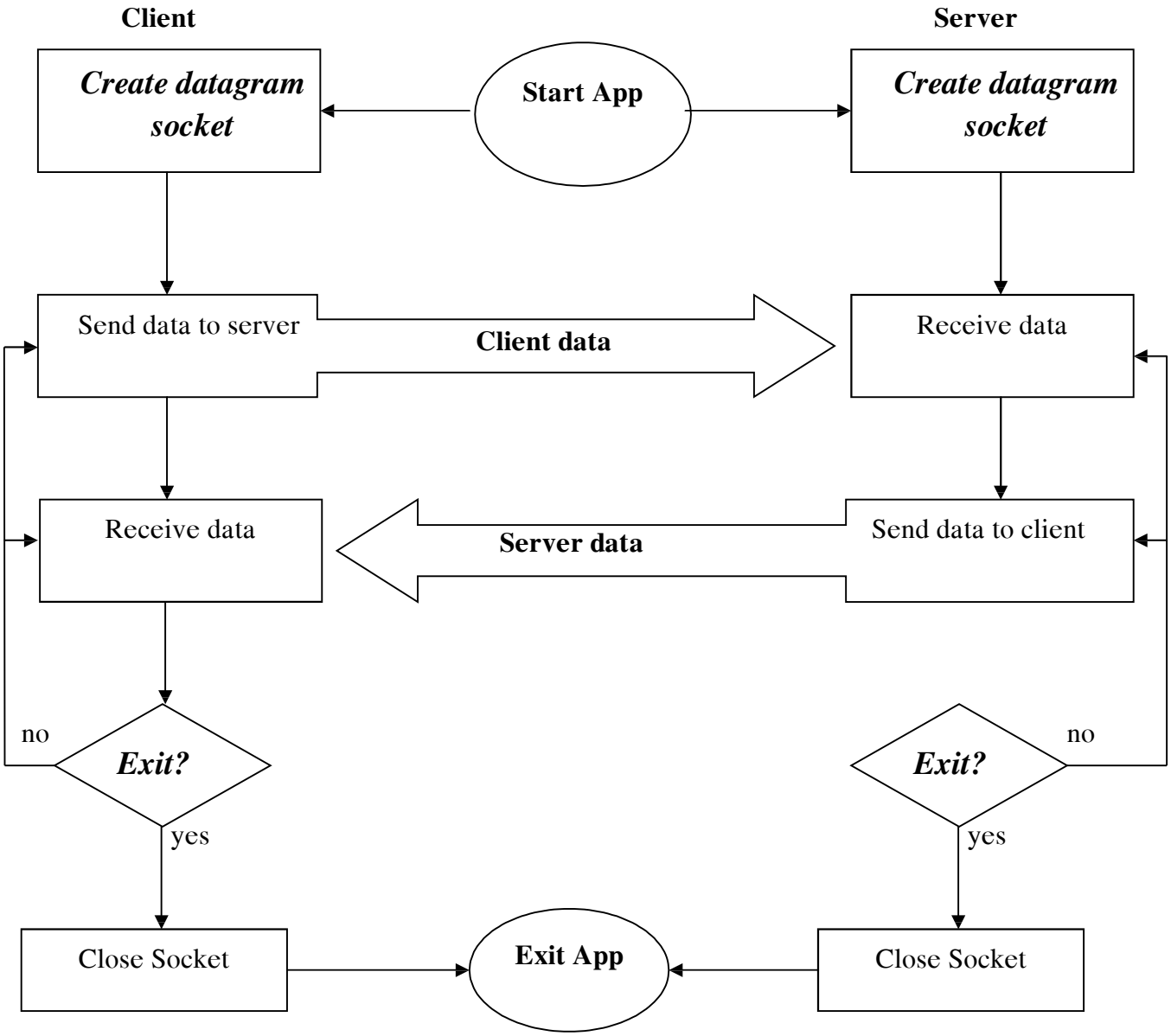
Chính vì những ưu điểm đó những nhà lập trình vẫn sử dụng UDP cho ứng dụng của họ, khi đó người ta dùng nhiều cách để xác nhận cho gói dữ liệu đi đến đích chính xác và trọn vẹn.

Một ví dụ minh họa cơ chế xác nhận:

1. Client gởi một gói dữ liệu(package) cho server và chờ đợi xác nhận từ server.
2. Server nhận được gói dữ liệu sẽ trả về thông điệp phản hồi cho client xác nhận gói dữ liệu đã nhận được.

Nếu client chờ đợi hơn một khoảng thời gian cho phép(time out) mà không nhận được phản hồi từ server thì nó cho là gói dữ liệu không đi đến đích và truyền lại, nếu sau nhiều lần không nhận được phản hồi từ server nó giả định rằng mối kết nối bị đứt hay server bị hỏng hóc.

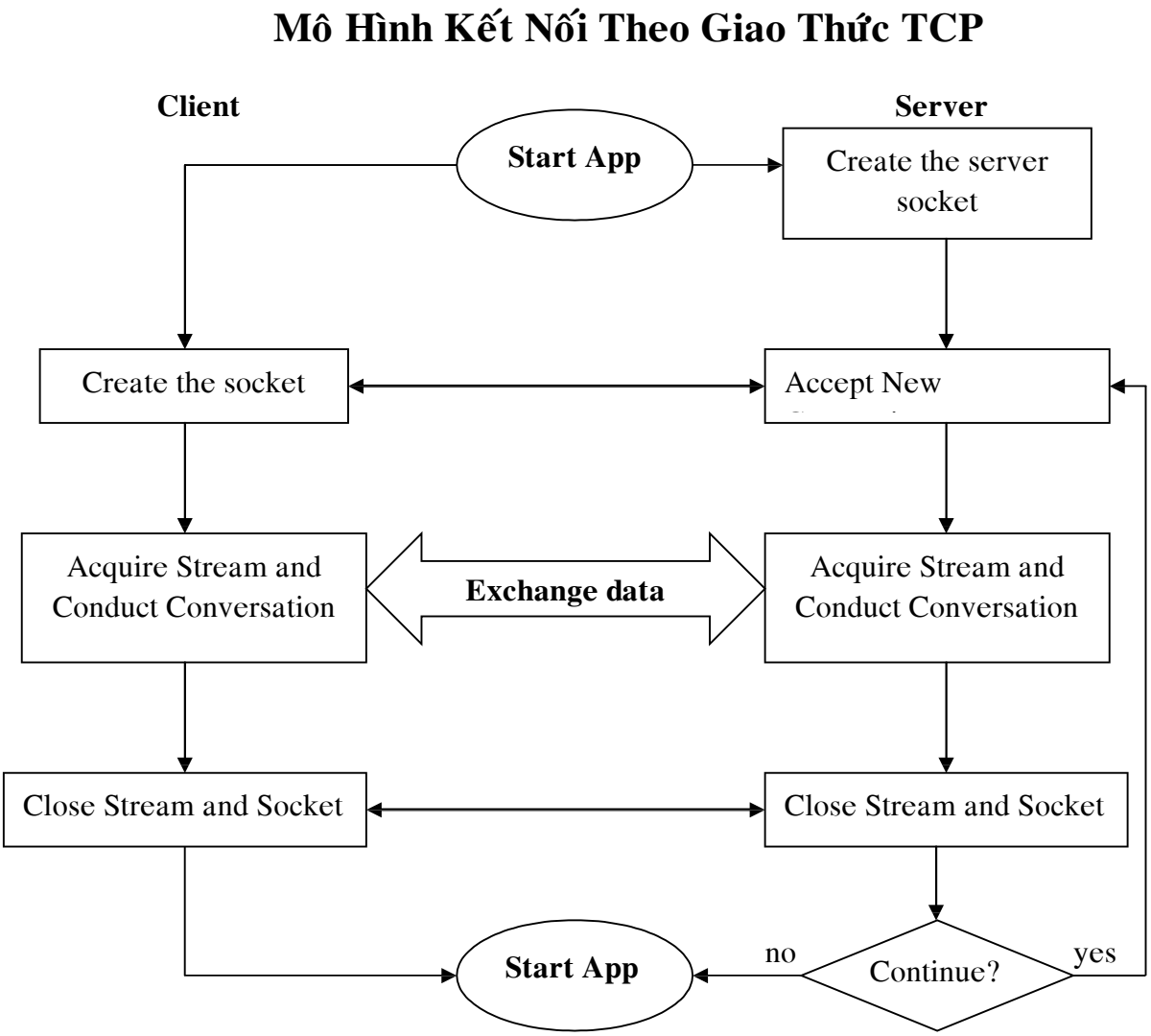
Mô Hình Kết Nối Theo Giao Thức UDP



Hình 12: *Mô hình kết nối theo giao thức UDP*

2.2 Giao thức TCP

TCP cung cấp khả năng truyền không lỗi từng gói dữ liệu gởi đi đến máy nhận theo giao thức giao thức này phải có trách nhiệm thông báo và kiểm tra xem dữ liệu có đến đủ hay chưa, có lỗi hay không có lỗi. Trước khi chuyển dữ liệu bao giờ cũng có việc thiết lập kênh truyền giữa hay máy. Do phải duy trì mối kết nối và kiểm tra dữ liệu nên sử dụng TCP phải đòi hỏi chiếm thêm một số tài nguyên và cách lập trình cho giao thức này hơi khó (phải thực hiện các bước kiểm tra dữ liệu theo yêu cầu của TCP). Truyền dữ liệu theo giao thức TCP thường áp dụng cho các dịch vụ như truyền tập tin, các dịch vụ trực tuyến trên Internet đòi hỏi có độ chính xác cao.



Hình 13: Mô hình kết nối theo giao thức TCP

3. Địa Chỉ IP

3.1 Giới thiệu địa chỉ IP

Tất cả các máy trong hệ thống mạng(LAN, WAN, Internet) đều có ít nhất 2 địa chỉ: địa chỉ vật lý(Mac Address) và địa chỉ Internet. Địa chỉ vật lý còn được gọi là Ethernet address là một dãy bit gồm 48 bit được gán bởi các nhà sản xuất, địa chỉ này được biểu diễn dưới dạng số thập lục phân (hecxa).

Ví dụ : 3A : 9D : 10 : 60 : 7C : 1F

Như thế mỗi card mạng(interface card) có một địa chỉ duy nhất địa chỉ này được quy định từ nhà sản xuất card mạng, tuy nhiên địa chỉ vật lý không thể hiện khả năng xác định vị trí của hệ thống trên mạng. Để giải quyết vấn đề đó người ta đưa ra địa chỉ IP(IP Address).

Địa chỉ IP phải là duy nhất trên mạng và có một dạng thống nhất, mỗi địa chỉ IP gồm có 4 byte và có 2 thành phần: địa chỉ đường mạng (Network ID) và địa chỉ host(Host ID).

<table>
<tr><td>Class ID</td><td>Network ID</td><td>Host ID</td></tr>
<tr><td colspan="3">32 bits(4 byte)</td></tr>
<tr><td colspan="3">Đia chỉ IP</td></tr>
</table>

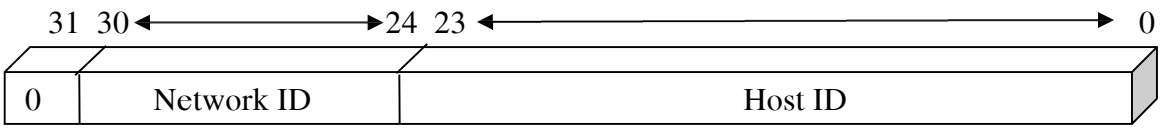
Nếu máy tính được nối mạng với Internet thì địa chỉ IP phải do NIC(Network Information Center) cấp.

3.2 Phân Loại Địa Chỉ IP

Có tất cả 5 lớp địa chỉ IP nhưng hiện nay có 3 lớp được sử dụng là lớp A, B, và C.

◆Lớp A:

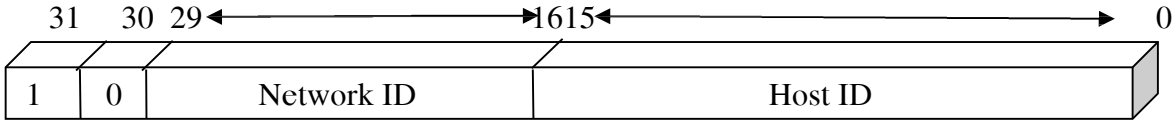
Dùng cho hệ thống mạng có số lượng địa chỉ host rất lớn, số lượng này có thể lên đến 16 triệu địa chỉ host.



Để có thể nhận biết địa chỉ thuộc lớp nào người ta căn cứ vào bit đầu tiên trong phần network ID, trong trường hợp lớp A: bit đầu tiên trong phần ID network bằng 0. 8 bits đầu dùng cho phần Network ID còn lại 24 bits dành cho phần Host ID. Như vậy có 126(2^7) địa chỉ đường mạng và 16.777.214 (2^{24})địa chỉ Host ID.

◆Lớp B:

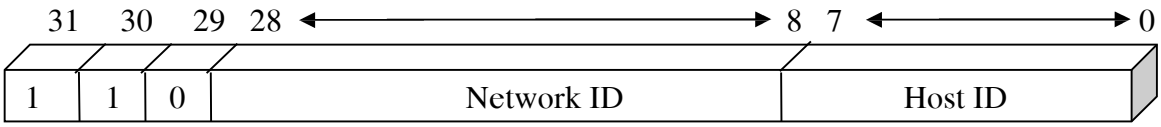
Dùng cho hệ thống mạng trung bình số lượng HostID lên đến khoảng 65 ngàn.



Địa chỉ lớp B được nhận biết qua bit đầu tiên trong phần Network ID bit đầu tiên có giá trị 1. Phần Network ID có 16 bit(2 byte) và phần Host ID có 16 bit như vậy số địa chỉ đường mạng trong lớp B này là 16.382(2^{14}-2)và số địa chỉ host 65.534(2^{16}-2).

◆**Lớp C:**

Địa chỉ lớp C dùng cho mạng nhỏ có số lượng máy không vượt quá 254 máy.



Có thể nhận biết địa chỉ lớp C thông qua 2 bit đầu tiên trong phần Network ID, hai bit này được bật lên 1. Phần NetWork ID có 2,097,150(2^{21}-2) địa chỉ đường mạng và phần Host ID có 254(2^{8}-2) địa chỉ host.

Chúng ta có thể xem số địa chỉ Host ID và Network ID qua bảng sau

Lớp Mạng	Số địa chỉ đường mạng	Số host trên một địa chỉ mạng
A	126	16,777,214
B	16,382	65,534
C	2,097,150	254

3.3. Subnet Mask(mặt nạ con)

Subnet mask là một dãy 32 bit giống như địa chỉ IP được dùng kèm với địa chỉ IP để xác định mạng con. Khi có một địa chỉ IP và kèm theo là một subnet mask chúng ta có thể xác định địa chỉ đường mạng con của địa chỉ IP đó bằng cách thực hiện toán tử AND giữa IP và subnet đây là cách mà router xác định cho gói dữ liệu đi theo đường mạng nào để để đến máy nhận.

Ví dụ : địa chỉ IP:192.125.125.3

Subnet mask :255.255.255.0

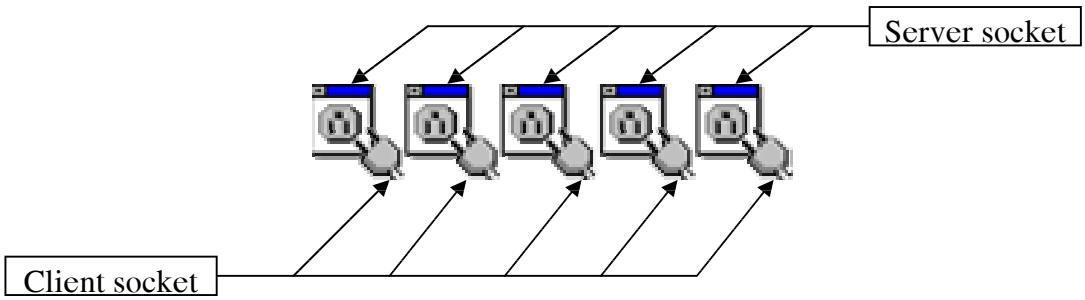
Chương II: Một số hàm socket

1. Khái niệm về socket

Trong mô hình mạng để hai máy tính có thể trao đổi thông tin cho nhau thì cần phải tạo ra kết nối giữa chúng. Trong quá trình làm việc người ta nhận thấy rằng những nhà lập trình ứng dụng rất khó khăn trong việc thiết lập kết nối và truyền tải dữ liệu giữa các máy tính với nhau. Vì thế người ta xây dựng khái niệm socket, khái niệm này được đưa ra đầu những năm 80 bởi các nhà khoa học máy tính ở California tại Berkeley. Khái niệm này được đưa ra từ ý tưởng

phân tầng, trong đó Windows sockets Application Programming Interface(Winsock API) là thư viện các hàm do hãng Berkeley Software Distribution of UNIX đưa ra. Nhằm làm đơn giản hóa quá trình thiết lập kết nối và chuyển dữ liệu. socket dựa trên giao thức TCP/IP tạo môi trường trung gian cho các ứng dụng và giao thức bên dưới.

Socket được xem là một cấu trúc dữ liệu trừu tượng(asbtraction data structure) dùng tạo ra một kênh truyền(channel) để gởi và nhận dữ liệu giữa các process trong cùng chương trình hay các giữa các máy trong cùng môi trường mạng với nhau. Hay nói một cách đơn giản hơn chúng ta xem socket như là "cơ chế ổ cắm". Khi kết nối giữa client và Server tương tự như việc cắm phích điện vào ổ cắm điện. client thường được xem như là phích cắm điện, còn server được xem như là ổ cắm điện, một ổ cắm có thể cắm vào đó nhiều phích điện khác nhau cũng như một server có thể phục vụ cho nhiều client khác nhau.



Hình 14: *minh họa cơ chế socket*

Trong quá trình truyền, nhận dữ liệu cần có một máy đóng vai trò là server và một máy đóng vai trò client, đầu tiên server phải tạo ra một socket và chờ đợi các yêu cầu kết nối từ client. client tạo ra socket cho riêng nó xác định vị trí server (dựa vào tên của server hay địa chỉ của server trong mạng) và tiến hành việc kết nối với server, sau khi kết nối được thiết lập client và server có thể tiến hành việc trao đổi dữ liệu với nhau.

2. Thư viện các hàm socket (API) trong Java.

Trong Java người ta cũng xây dựng các lớp về socket phục vụ cho việc truyền tải dữ liệu dể dàng và nhanh chóng, các lớp này được đóng gói trong gói Java.net.

Một số lớp cần thiết trong gói Java.net

2.1 Lớp InetAddress

Vì địa chỉ Internet theo số IP và theo tên rất thường dùng khi kết nối vào mạng cho nên Java xây dựng hẳn một lớp InetAddress dành riêng cho việc quản lý địa chỉ theo tên và số lớp. Lớp InetAddress cung cấp các phương thức static thông dụng nhất dùng để chuyển đổi và truy xuất địa chỉ IP (không có phương thức khởi dựng cho lớp này). Thường ta sẽ quan tâm đến các phương thức sau:

public static InetAddress getLocalHost() throws UnknownHostExceptiongetByName

Trả về đối tượng InetAddress là địa chỉ máy cục bộ(local host).

public static InetAddress getByName(String host) throws UnknownHostException

phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi và trả về đối tượng InetAddress thay mặt cho địa chỉ máy này.

public static InetAddress[] getAllByName(String host) throws UnknownHostException

phương thức này nhận địa chỉ của một máy bằng kiểu chuỗi và trả về tất cả đối tượng InetAddress thay mặt cho địa chỉ máy này.

Public byte[] getAddress()

Trả về địa chỉ IP của đối tượng InetAddress dưới dạng một dãy các byte. Vị trí byte cao nhất nằm ở byte 0.

Public String getHostAddress()

Trả về địa chỉ IP của đối tượng InetAddress dưới dạng một chuỗi được định dạng phân thành làm 4 nhóm %d.%d.%d.%d (ví dụ "127.16.11.12").

2.2 Lớp Socket

Lớp Socket dùng tạo kết nối từ phía máy khách với máy chủ trương được khởi dựng bằng các phương thức sau:

Public Socket(String host, int port)
Throws UnknownHostException, IOException

Tạo ra một socket kết nối theo địa chỉ host và số hiệu cổng port.

Public Socket(InetAddress address, int port) Throws IOException

Tạo ra một Socket kết nối từ địa chỉ là đối tượng InetAddress và số cổng port.

Public Socket(String host, int port, boolean stream) throws IOException.

Tạo ra một socket kết nối theo địa chỉ host và số cổng port, tham số stream cuối cùng để quy định kết nối theo TCP(stream=true)hayUDP(stream=false). Tuy nhiên nếu áp dụng để tạo socket cho giao thức UDP nên sử dụng lớp thay thế là DatagramSocket.

Các phương thức khác hỗ trợ cho lớp Socket từ phía máy khách bao gồm:

InputStream getInputStream() Throws IOException

Lấy về luồng nhập để máy khách có thể đọc dữ liệu trả về từ phía máy chủ.

OutputStream getOutputStream() throws IOException

Lấy về luồng xuất để máy khách có thể ghi dữ liệu gửi đến máy chủ.

InetAddress getInetAddress()

Lấy địa chỉ kết nối socket của máy chủ

Int getPort()

Lấy về số cổng dùng kết nối của máy chủ.

Synchronized void close() throws IOException

Cắt đứt kết nối với máy chủ.

Ví dụ sau sẽ thực hiện kết nối với máy chủ có địa chỉ "localhost"và mở 2 luồng xuất nhập để đọc và gửi thông tin đến máy chủ theo cổng 1234;

```
try{
        Socket me=new Socket("localhost",1234);
        DataInputStream in= new DataInputStream(me.getInputStream());
        DataOutputStream out= new DataOutputStream(me.getOutputStream());
}catch(Exception e)
{
        System.out.println(e);
}
```

2.3 Lớp ServerSocket

Lớp ServerSocket dùng tạo kết nối máy chủ với máy khách. Đối tượng ServerSocket được tạo ra trên máy chủ và lắng nghe những kết nối từ phía máy khách gửi đến theo một số cổng định trước. Đối tượng ServerSocket được khởi dựng từ phương thức sau:

Public ServerSocket(int port) throws IOException

Port là số hiệu cổng mà đối tượng ServerSocket phải lắng nghe để nhận biết những kết nối từ phía máy khách gởi đến.

Để chờ đợi kết nối từ các máy khác gửi đến đối tượng ServerSocket thường nhờ đến phương thức accept như sau:

Socket accept() throws IOException

Phương thức này thực sự dừng lại chờ đợi cho đến khi nhận được thông tin kết nối sẽ trả về đối tượng socket của máy khách nơi có yêu cầu nối vào máy chủ.

Cuối cùng máy chủ có thể cắt đứt mọi kết nối bằng cách gọi phương thức close của đối tượng serversocket:

Public void close() throws IOException

Ví dụ đoạn mã sau sẽ tạo một đối tượng ServerSocket trên máy chủ luôn lắng nghe kết nối từ máy khách gởi đến qua số cổng 1234.

```
Try{
        ServerSocket server=new ServerSocket(1234);
        Socket client;
        //Chương trình sẽ dừng lại ở đây để chờ đợi sự kết nối
        client=server.accept();
        //có một kết nối gởi đến từ máy khách
        system.out.println("accept connection");
        //xử lý yêu cầu dịch vụ
        //....
        //cắt đứt các kết nối
        client.close();
        server.close();
}catch(Exception e)
{
        System.out.println(e);
}
```

2.4 Lớp DatagramSocket

Lớp này được dùng để chuyển đi một gói dữ liệu (biểu diễn bằng đối tượng DatagramPackage) theo giao thức UDP. Dữ liệu được gửi đi không an toàn có thể bị lỗi trên đường truyền. Dưới đây là một số phương thức thường dùng của lớp DatagramSocket.

Public DatagramSocket() throws SocketException

Phương thức khởi dựng để tạo kết nối UDP.

Public DatagramSocket(int port) throws SocketException

Phương thức khởi dựng để tạo kết nối UDP với số hiệu cổng port.

Public void synchronized send(DatagramPackage p) throws IOException

Gói dữ liệu đi.

Public void synchronized receive(DatagramPackage p) throws IOException

Nhận gói dữ liệu về.

Public void synchronize close();

Đóng kết nối.

2.5 Lớp DatagramPackage

Lớp này dùng cho một gói chứa dữ liệu gửi đi trên mạng theo kết nối DatagramSocket. Một gói có thể chứa thông tin như chiều dài gói, các địa chỉ IP và số cổng mà từ đó gói dữ liệu được chuyển đi. Dưới đây là một số phương thức hữu dụng của lớp DatagramPackage.

Public DatagramPackage(byte buf[], int len)

Phương thức khởi dựng gói có dữ liệu chứa trong bộ đệm buff[], chiều dài gói là len.

Public DatagramPackage(byte buf[], int len, InetAddress iaddr, int port)

Phương thức khởi dựng gói có dữ liệu chứa trong bộ đệm buff[], chiều dài gói là len, địa chỉ máy đích, và số hiệu cổng.

Public InetAddress getAddress()

Trả về địa chỉ IP chứa trong gói dữ liệu

Public byte[] getData()

Trả về dữ liệu thật sự chứa trong gói.

Public int getLength()

Trả về kích thước hay chiều dài gói dữ liệu.

Public int getPort()

Trả về số hiệu cổng chứa trong gói dữ liệu.

3. Chương trình minh họa cho việc sử dụng socket trong Java

3.1 Chương trình hoạt động theo giao thức TCP

Chương trình hoạt động dựa trên mô hình client/server, phải có 2 chương trình, chương trình thứ nhất đóng vai trò là server chạy trên máy chủ lắng nghe kết nối từ client. Chương trình thứ 2 client chạy trên máy khách kết nối đến máy chủ.

3.1.1 Chương trình client chạy trên máy khách

```
//chương trình client chạy trên máy khách
import java.io.*;
import java.net.*;
public class EchoClient {
   public static void main(String[] args) throws IOException {
  Socket echoSocket = null;
   PrintWriter out = null;
   BufferedReader in = null;
  try
        {
              echoSocket = new Socket("localhost",8080);
              out = new PrintWriter(echoSocket.getOutputStream(), true);
              in = new BufferedReader(new
InputStreamReader(echoSocket.getInputStream()));
        }
        catch (IOException e)
```

```
        {
                System.out.println("Error: Khong mo duoc socket " + e);
                System.exit(1);
        }
  BufferedReader stdIn = new BufferedReader(
                                new InputStreamReader(System.in));
   String userInput;
   while ((userInput = stdIn.readLine()) != null)
        {
                out.println(userInput);
                if (userInput.equals("Bye")) {
                System.out.println("Connection closed");
                break;
        }
        System.out.println(in.readLine());
   }
   out.close();
   in.close();
   stdIn.close();
   echoSocket.close();
  }
}
```

3.1.2 Chương trình server chạy trên máy chủ

```
//chương trình server chạy trên máy chủ
import java.io.*;
import java.net.*;
public class EchoServer {
   public static void main(String[] args) throws IOException {
        ServerSocketrSocket serverSocket = null;
        Socket clientSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
           serverSocket = new ServerSocket(6667);
        } catch (IOException e) {
           System.out.println("Khong tao duoc socket tren cong 7000");
           System.exit(-1);
        }
        try {
           clientSocket = serverSocket.accept();
        } catch (IOException e) {
           System.out.println("Error: khong mo duoc cong 7000");
           System.exit(-1);
```

```
        }
        out = new PrintWriter( clientSocket.getOutputStream(), true);
        in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        String inputLine = null;
        while ((inputLine = in.readLine()) != null) {
           out.println("[" + inputLine + "] accepted. Type \"Bye\" to end connection");
           System.out.println("Accepted: " + inputLine);
           if (inputLine.equals("Bye"))
              break;
        }
        out.close();
        in.close();
        clientSocket.close();
        serverSocket.close();
    }
}
```

3.2 Chương trình hoạt động theo giao thức UDP

cũng tương tự như giao thức TCP gồm 2 chương trình client và server

3.2.1 Chương trình client chạy trên máy khách

```
//chương trình ExchangeClient
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
public class ExchangRateClient{
        public static void main(String[] args)
        {
                // Tao cua so cho chuong trinh chinh
                Frame myWindow = new Frame("Stock Exchange Application");
                // Vung van ban dung de thong bao ty gia
                TextArea rateTable = new TextArea ("Wait. .. ");
                Label rateLabel = new Label("Exchane Rate Table");
                // Dat vi tri cho cac doi tuong
                rateTable.setBounds(new Rectangle(16, 33, 240, 100));
                rateLabel.setBounds(new Rectangle(16, 6, 158, 21));
                myWindow.setLayout(null);
                myWindow.add(rateTable, null);
                myWindow.add(rateLabel, null);
                // Xu ly tinh huong cho cua so chuong trinh chinh
                myWindow.addWindowListener(new WindowAdapter()
```

```
                {
                        public void windowClosing(WindowEvent event)
                        {
                                System.exit(1);
                        }
                }
                );
                myWindow.setSize(new Dimension(300, 150));
                myWindow.show();
                //Tao phan tuyen dung de cap nhat ty gia sau moi giay
                ExchangeThread exRate = new ExchangeThread(rateTable);
                exRate.start();
        }
}
//  Cai dat phan tuyen chay song song dung de cap nhat ty gia
class ExchangeThread extends Thread
{
        TextArea rateTable;
        // Tao doi tuong dung de trao doi ty gia voi may chu
        ExchangeData rate = new ExchangeData();
        // Phuong thuc khoi dung se luu lai bang ty gia do chuong trinh chinh dua sang
        public ExchangeThread       (TextArea rateTable)
        {
                this.rateTable = rateTable;
        }
        public void run()
        {
                while (true)
                {
                        String data = rate.getRates();
                        rateTable.setText(data);
                        delay(100);
                }
        }
        // Phuong thuc nay dung de tri hoan mot khoang thoi gian cua phan tuyen
        private void delay(int miliSeconds)
        {
                try
                {
                        this.sleep(miliSeconds);
                }
                catch(Exception e)
                {
```

```
                        System.out.println("Sleep error !");
                }
        }
}
// Thiet ke lop ExchangeData chiu trach nhiem gui va nhan so lieu tra ve tu may chu
class ExchangeData
{
        DatagramSocket socket;
        InetAddress serverAddress;
        String localHost;
        int bufferLength = 256;
        byte inBuffer[] = new byte[bufferLength];
        byte outBuffer[];
        DatagramPacket outDatagram;
        DatagramPacket inDatagram;
        public ExchangeData()
        {
                try
                {
                        // Tao doi tuong ket noi socket theo giao thuc UDP
                        socket = new DatagramSocket();
                        //Tao goi data dung de nhan ve
                        inDatagram = new DatagramPacket(inBuffer, inBuffer.length);
                        // Lay ve doi tuong InetAddress cho biet chi tiet dia chi may chu
                        serverAddress=InetAddress.getByName("localhost");
                }
                catch(Exception e){
                        System.out.println("Connect Error !");
        }
}
// Phuong tuc nay dung de chuyen yeu cau va nhan du lieu tra ve tu may chu
public String getRates()
{
        String data="";
        try
        {
                // Tao vung dem va gui yeu cau nhan ty gia den may chu
                outBuffer = new byte [bufferLength];
                outBuffer = "rate".getBytes();
                outDatagram = new DatagramPacket(outBuffer, outBuffer.length, serverAddress,
2345);
                socket.send(outDatagram);
                // Cho nhan ket qua tra ve tu may chu
```

```
                socket.receive(inDatagram);
                // Lay thong tin tu goi du lieu nhan duoc
                InetAddress destAddress = inDatagram.getAddress();
                String destHost = destAddress.getHostName().trim();
                int destPort= inDatagram.getPort();
                // Lay du lieu that su chua trong goi
                data = new String(inDatagram.getData());
                data = data.trim();
        }
        catch(Exception e)
        {
                System.out.println("IO Exception Occurreded !");
        }
        return data;
}
}
```

3.2.2 Chương trình server chạy trên máy chủ

```
//chương trình ExchangeServer
import java.net.*;
import java.io.*;
import java.util.*;
public class ExchangRateServer{
        private static String getNewYorkRate(){
                return Double.toString(Math.random()*135);
        }
        private static String getHongKongRate(){
                return Double.toString(Math.random()*135);
        }
        private static String getTokyoRate(){
                return Double.toString(Math.random()*135);
        }
        public static void main (String args[]){
                try{
                //Tao doi tuong ket noi socket theo giao thuc UDP tai cong 2345
                        DatagramSocket socket=new DatagramSocket(2345);
                //Lay dia chi va so cong ket noi cua may chu
                        String localAddress=
                        InetAddress.getLocalHost().getHostName().trim();
```

```
                    int localPort= socket.getLocalPort();
                    //In cac thong so ra man hinh
                    System.out.print(localAddress+":");
                    System.out.println("Exchange Rate is listening on port "
                    +localPort+".");
                    //Tao bo dem dung de goi va nhan du lieu do may khach goi den
                    int bufferLength=256;
                    byte outBuffer[];
                    byte inBuffer[]=new byte[bufferLength];
                    //Tao goi du lieu de goi di
                    DatagramPacket outDatagram;
                    //Tao goi du lieu de nhan ve
                    DatagramPacket inDatagram=new
DatagramPacket(inBuffer,inBuffer.length);
                    //Vong lap cho nhan du lieu do may khach goi den
                    boolean finished=false;
                    do{
                            //cho nhan du lieu
                            socket.receive(inDatagram);
                            InetAddress destAddress= inDatagram.getAddress();
                            String destHost= destAddress.getHostName().trim();
                            int destPort= inDatagram.getPort();
                            System.out.println("\nReceive datagram from "+destHost+" at port
"+destPort);
                            String data = new String (inDatagram.getData()).trim();
                            System.out.println("It container data: "+data);
                            //Cham dut khi nhan duoc tu "quit"
                            if (data.equalsIgnoreCase("quit")) finished=true;
                            //Lay cac thong tin ve ti gia
                            String s= new Date().toString();
                            s=s+"\n NewYork: "+getNewYorkRate();
                            s=s+"\n Tokyo: "+getTokyoRate();
                            s=s+"\n HongKong: "+getHongKongRate();
                            //Goi thong bao ti gia den may khach
                            outBuffer=s.getBytes();
                            outDatagram=new DatagramPacket(outBuffer,

        outBuffer.length,destAddress,destPort);
                            socket.send(outDatagram);
                            System.out.println("Send "+s+" to "+ destHost+" at port"+destPort);
                    }while (!finished);
            }catch(IOException ex){
                    System.out.print("IOException occurred")
```

Phần III:
Xử Lý Đa Tiến Trình(multitasking) và Đa Luồng(multithreading)

- ✓Chương I: Đa Tiến Trình(multitasking)
- ✓Chương II: Đa Luồng(multithreading)

Chương I: Đa Tiến Trình (multitasking)

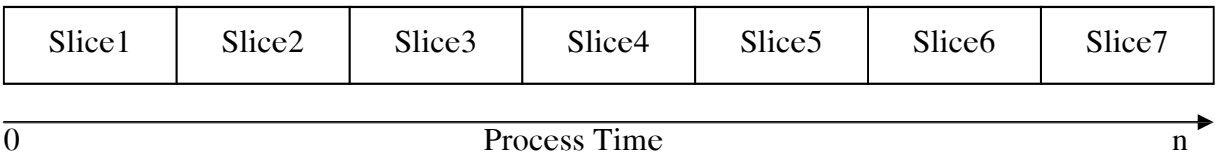
Lúc ban đầu, lập trình viên không có hệ điều hành mà giao tiếp trực tiếp với máy tính. Sao đó, lập trình viên xử lý tập tin batch đơn giản trên hệ điều hành hỗ trợ thực thi những tác vụ đơn lẻ. Ở các hệ thống Single-tasking, một khi tác vụ (process) được khởi động thì nó sẽ chạy hoàn tất trước khi tác vụ khác có thể được khởi động, ví dụ như hệ điều hành DOS.

Để có thể thực thi một chương trình trên hệ điều hành single-tasking thường phải tốn nhiều thời gian chờ đợi cho những tác vụ có thời gian xữ lý lâu như I/O, không phát huy tối đa khả năng CPU(vì phải chờ những tác vụ I/O).

Để giải quyết vấn đề người ta đưa ra hệ điều hành multitasking.

Như vậy multitasking được định nghĩa là việc thi hành đồng thời 2 hay nhiều tác vụ trên một CPU.

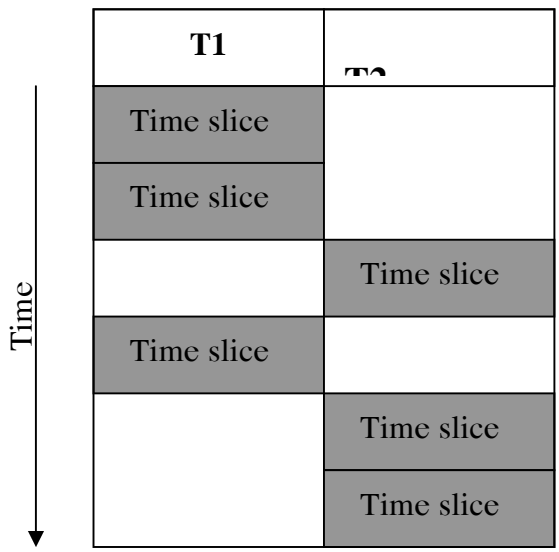
Nhiều tác vụ khởi động để chạy trên một CPU. Hệ điều hành có trách nhiệm chuyển đổi các tác vụ thực thi trên CPU. Cách thức hệ điều hành điều khiển thi hành đồng thời nhiều tác vụ bằng cách gán cho CPU một tác vụ ở một thời điểm xác định. Hệ điều hành multitasking tạo nên ảo giác thi hành đồng thời bằng cách chia thành nhiều múi thời gian(time-slice).



Khi có nhiều tác vụ thi hành cùng một lúc, mỗi tác vụ được CPU phục vụ trong một số lượng múi thời gian nhất định. Như vậy thực sự trong một thời điểm CPU chỉ phục vụ cho một tác vụ duy nhất, nhưng khoảng thời gian chuyển xử lý giữa các tiến trình rất nhỏ nên ta có cảm giác chúng được thi hành đồng thời.

Ví dụ:

T1,T2 là 2 tiến trình xử lý đồng thời.



Tiến trình ưu tiên cho một tác vụ chạy sau một khoảng thời gian thực thi tác vụ khác được gọi là chuyển đổi ngữ cảnh(context switching).

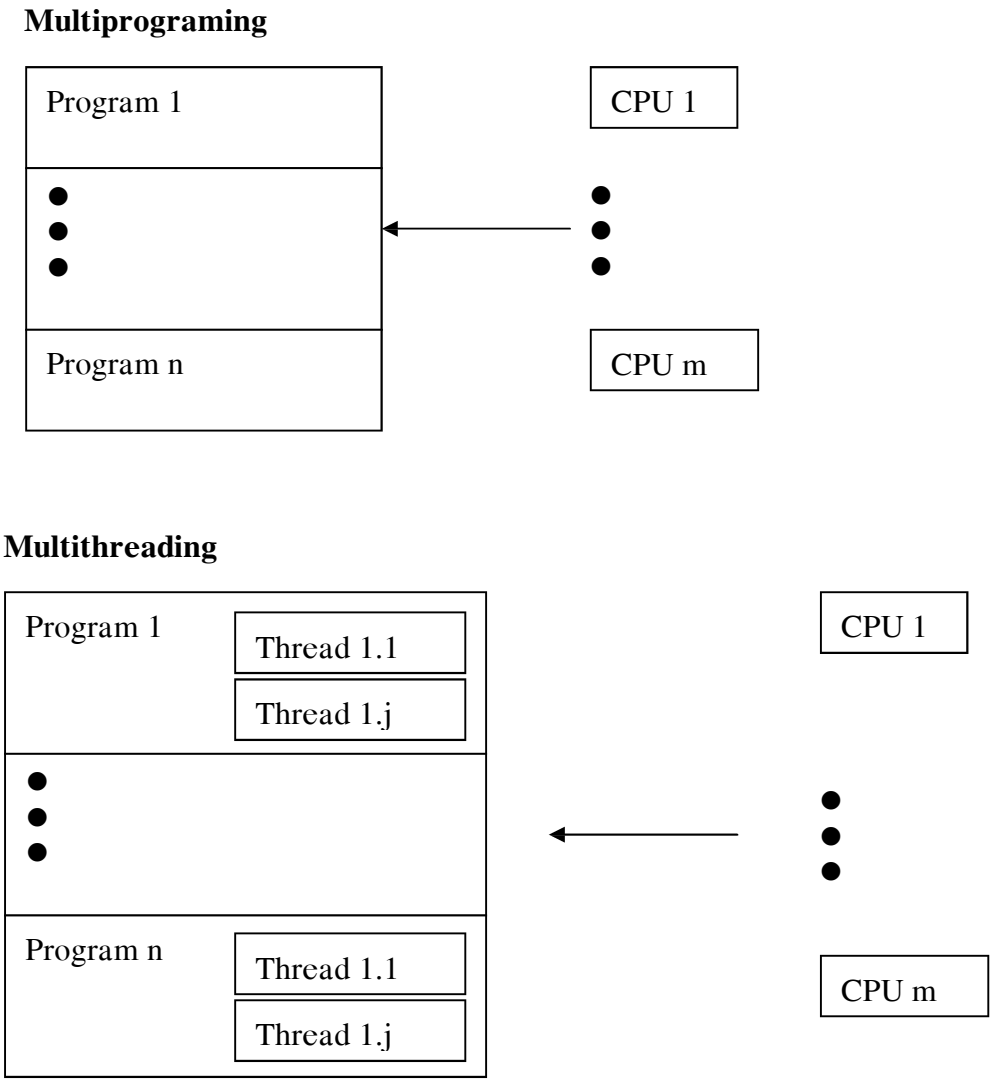
Các hệ điều hành multitasking có thể là ưu tiên (preemptive) hoặc không ưu tiên (nonpreemptive). Ở trường hợp hệ điều hành preemptive, ứng dụng không cần biết sự chuyển đổi giữa các tiến trình (sự chuyển đổi này được thi hành bởi hệ điều hành), sự khác nhau giữa hệ điều hành preemptive và nonpreemptive: ở hệ điều hành nonpreemptive là ứng dụng có nhiệm vụ từ bỏ CPU. Dạng multitasking cũng được tham khảo đến như là cooperative multitasking. Các hệ điều hành như Netware và windows 3.x là nonpreemptive multitasking, còn các hệ điều hành khác như MVS, VMS, UNIX,MAC, NT, và OS/2 là các hệ điều hành preeptive multitasking thật sự.

Các tác vụ và tiến trình là các khái niệm cơ bản của bất kì hệ điều hành nào. Hệ điều hành phải có chức năng tạo hủy các tiến trình.

Chương II: Đa Luồng(multithreading)

1. Khái niệm luồng

Các chương trình truyền thống thực thi theo một kiểu tuần tự với một dòng (luồng,thread) điều khiển đơn độc, một sự nối tiếp các lệnh được thực thi bởi một tiến trình, một tiến trình nhiều hơn một luồng điều khiển được gọi là một tiến trình đa luồng (multithreaded processor). Một thread là một luồng điều khiển tuần tự đơn trong một chương trình, nó là sự nối tiếp các lệnh được thực thi trong một tiến trình.



Hình 15: *Đa luồng(multithreading) khác với đa chương(multiprograming)*

Mỗi thread có một điểm thực thi riêng lẻ. Các thread thường tham khảo đến như là các thread thực thi (thread execution), bởi vì các thread trong một tiến trình là kết hợp những lệnh nối tiếp nhau. Trong một chương trình đa luồng có thể có nhiều thread chạy đồng thời trong một

không gian địa chỉ, mỗi thread có thể được xem như một processor ảo với bộ đếm chương trình(process counter),stack và tập thanh ghi riêng nó. Các thread là đơn vị cơ bản của sự thực thi sử dụng CPU.

Mỗi thread trong một tiến trình chạy độc lập với các thread khác. Tất cả các thread trong một tiến trình chia sẻ một không gian địa chỉ chung và có quyền truy xuất ngang nhau đến tất cả các tài nguyên của tiến trình. Vì thread chia sẻ chung vùng không gian địa chỉ nên hành động của một thread có thể ảnh hưởng đến những thread khác trong một tiến trình.

Khái niệm về thread và process là tương tự, một process có quyền sở hữu tài nguyên(thí dụ: memory, mở file,...), trong khi các thread là đơn vị có thể ra lệnh làm việc. Hầu hết các hệ điều hành multithread định thời các thread chạy trên một CPU. Nhiều thread không cần thiết chạy song song. Trên một đơn xử lý các thread được chia múi thời gian bởi hệ điều hành trong khi trên các máy có nhiều bộ xử lý các thread chạy song song trên nhiều bộ vi xử lý khác nhau.

Thread hỗ trợ lập trình đồng thời và thường được dùng cho các tác vụ song song trong một trình ứng dụng. Các tiến trình quan trọng được thi hành song song trong một ứng dụng. Thread dễ tạo hơn là các tiến trình, và việc chuyển đổi ngữ cảnh cũng nhanh hơn các tiến trình. Vì việc duy trì ngữ cảnh của thread cũng nhẹ hơn nhiều so với process, cho nên thread còn được gọi là LightWeight Processes(LWP).

Một việc rất quan trọng cần nhớ là một ứng dụng với rất nhiều công việc cần thực hiện mà nó chỉ có một thread sẽ chạy chậm hơn so với các máy có nhiều bộ xử lý cho đến khi ứng dụng được chia thành nhiều thread để thực thi. Một thread trong một tiến trình có thể chạy trên bất kỳ bộ xử lý nào và vì thế có khả năng khai thác tính song song vốn có của các máy có nhiều bộ xử lý.

2. Những tiện ích khi dùng thread (Advantages of multithreading)

Nếu dùng thread một cách đúng đắng thì thread có các tiện lợi sau:

➢ Tăng thông lượng và hiệu năng tốt hơn.

Các chương trình Multithreaded có thể thực thi trên môi trường Multiprocessor(MP). Trong môi trường MP, các thread như là tiến trình thực thi song song trên nhiều CPU như vậy thông lượng và sự thực thi tốt hơn. Thread có thể khai thác khả năng song song vốn có của các máy có nhiều bộ xử lý như vậy sẽ rút ngắn thời gian hoàn thành công việc.

Còn các máy đơn xử lý, các thread được cung cấp thông lượng tốt hơn bởi vì CPU có thể xử lý nhiều thread đồng thời khi có một vài thread bị block trên tác vụ I/O. Ví dụ một tiến trình có 2 thread thực thi trên máy đơn xử lý, khi một thread bị block trong khi gọi một hàm hệ thống (ví dụ file I/O), thì thread còn lại vẫn tiếp tục chạy.

➢ Thuật giải đơn giản

Nhiều tác vụ chương trình có thể mã hóa tốt hơn bởi các giải thuật trong các thành phần nhỏ và được phân cho các thread xử lý các thành phần này. Các thread cải tiến tốt hơn cho thiết kế chương trình

➢ Tính phản hồi cao(More responsive programs):

Bên cạnh những thread đơn, giao diện lập trình với người sử dụng được tính toán trong một thời gian dài, trong khi tiến hành sự tính toán, chương trình không thể tương tác với người sử dụng, bởi vì sử dụng các thread đơn để tính toán, thread chính hay GUI thread có thể thuận lợi cho người sử dụng.

➢ Tăng tính song song (Cheaper concurrency model):

Trong mô hình lập trình client – server, các chương trình server có thể xuất hiện như một thread xử lý đồng thời cho mỗi client.

3. Các khó khăn khi dùng thread

- Added complexity(phức tạp).
- Difficult to debug and test(khó kiểm tra và debug).
- Data synchronization and race condition(sự đồng bộ dữ liệu và các điều kiện tranh đua).
- Potential for deadlock(khả năng bị deadlock).
- Non – thread – safe environment(môi trường thread không an toàn).

4. Mô hình tiểu trình(thread) trong JAVA

Những khái niệm về thread trong java cũng giống như các khái niệm được nêu ở trên.

Hệ thống Java chạy dựa trên các thread và các lớp thư viện thiết kế với chức năng multithreading, Java sử dụng hiệu quả các tiểu trình này ngay trong môi trường không đồng bộ. Điều này làm giảm thiểu sự lãng phí CPU.

Để tạo ra thread trong Java có 2 cách:

✓ Tạo lớp dẫn xuất từ lớp **thread** của java.

✓ Cài đặt giao tiếp **Runnable.**

a. Ta chỉ cần khai báo như ví dụ sau:

Import java.lang.Thread

Public class GreatClass extends Thread

b. Khi cài đặt giao tiếp Runnable, ta cần phải khai báo phương thức run() trong lớp đang xét. Phương thức Run là phương thức cần có của giao tiếp Runnnable. Trong phương thức run(), ta thực hiện tất cả các việc phải làm của từng thread.

public class Great extends java.applet.Applet
implements Runnable

sau đó ta cần khai báo một đối tượng thread như là một vùng dữ liệu của lớp.
Khởi tạo đối tượng Thread và cho thực hiện thread bằng phương thức start().
Chấm dứt một thread bằng cách gọi phương thức stop().
Ví dụ cài đặt theo phương thức Runnable:

```
/*
// header - edit "Data/yourJavaHeader" to customize
// contents - edit "EventHandlers/Java file/onCreate" to customize
//
*/
import java.awt.*;
import java.applet.*;
public class ThreadApplet extends Applet implements Runnable
{
```

```
        //khai bao bien doi tuong thread
        Thread thread;
        int count;
        String displayStr;
        Font font;
        //khai bao phuong thuc start
        public void start()
        {
                font = new Font("TimeRoman",Font.PLAIN,72);
                setFont(font);
                count=0;
                displayStr="";
                thread= new Thread(this);
                thread.start();
        }
        //khai bao phuong thuc stop
        public void stop()
        {
                thread.stop();
        }
        //khai bao phuong thuc run
        public void run()
{
                while(count<1000)
                {
                        ++count;
                        displayStr= String.valueOf(count);
                        repaint();
                        try
                        {
                                //cho biet thread tam nghi trong 100ms
                                thread.sleep(1000);

                        }
                        catch(InterruptedException e)
                        {
                        }
                }
        }
        //khai bao phuong thuc paint
        public void paint(Graphics g)
        {
```

```
            g.drawString(displayStr,50,130);
        }
}
```

5. Tính chất thread.

Java cho mỗi thread một độ ưu tiên trong tất cả các thread đang xử lý. Độ ưu tiên là một số nguyên cho biết thứ tự ưu tiên của nó với các thread khác. Độ ưu tiên của thread dùng để quyết định khi nào có thể chuyển sang thực hiện thread kế tiếp. Đây được gọi là chuyển đổi ngữ cảnh (context switch).

Trong trường hợp 2 thread có cùng độ ưu tiên tranh giành CPU. Với hệ điều hành như windows 98 các thread có cùng độ ưu tiên được phân chia tự động. Với hệ điều hành khác như Solaris 2.x, các thread cùng cấp phải tự động nhường điều khiển cho thread khác. Nếu không làm điều này các thread khác sẽ không được chạy.

6. Đồng bộ hóa các thread

Vì multithreading xử lý công việc không đồng bộ nên phải có cách đồng bộ hóa khi cần thiết. Ví dụ nếu bạn muốn hai thread liên kết và phân chia một cấu trúc dữ liệu phức tạp như danh sách liên kết, bạn cần vài cách chắc rằng chúng không đụng độ nhau. Bạn phải ngăn cản một thread đang ghi dữ liệu trong khi một thread khác đọc dữ liệu đó. Để thực điều này Java dùng kỹ thuật *monitor*. Monitor do C.A.R. Hoare đưa ra đầu tiên. Bạn có thể xem monitor là chiếc hộp nhỏ có thể giữ một thread. Một thread được nạp vào một monitor, tất cả các thread khác phải đợi cho đến khi thread đó thoát ra khỏi monitor.

Hầu hết các hệ thống đa tiểu trình xem monitor như những đối tượng mà chương trình phải giành được. Trong Java không có lớp "Monitor", mà có các đối tượng ẩn monitor được tự động tạo ra khi phương thức đồng bộ hóa được gọi. Khi một thread đã ở trong một phương thức đồng bộ, không có thread nào khác có thể gọi phương thức đồng bộ khác trong cùng một đối tượng. Điều này cho phép lập trình thread rất đơn giản và trong sáng.

7. Các phương thức đồng bộ(synchronized)

Đồng bộ hóa rất dể dàng trong Java vì các đối tượng đều có monitor đi kèm. Để truy xuất monitor của đối tượng chỉ cần gọi phương thức có thêm từ khóa **synchronized.** Trong khi một thread đang ở trong phương thức đồng bộ hóa, tất cả các thread khác đang chờ cố gắn gọi phương thức này. Để thoát khỏi monitor và từ bỏ điều khiển của một đối tượng để nhận tiểu trình kế tiếp đang chờ, monitor dừng phương thức đồng bộ.

Để hiểu sự cần thiết của việc đồng bộ hóa hãy bắt đầu với ví dụ đơn giản không cần đến việc đồng bộ này – nhưng việc đồng bộ sẽ sử dụng. Chương trình sau có 3 lớp đơn giản, lớp đầu tiên là Callme với phương thức call(). Phương thức call() nhận đối số **msg** kiểu **String**, sau đó in chuỗi **msg** trong dấu ngoặt vuông. Điều thú vị là sau khi call() in chuỗi **msg** nó gọi Thread.sleep(1000) để ngưng thread hiện tại trong một giây.

Lớp kế tiếp là **Caller** với cấu tử nhận đối số tham chiếu đến lớp **Callme** và kiểu **String.** Cấu tử cũng tạo một thread mới thông qua phương thức **run()** và phương thức **run()** của lớp này gọi phương thức **call()** trên đối số **msg** của **Callme**. Sau cùng là lớp **Synch** khởi động bằng cách gọi của lớp **Callme** và ba cách gọi của lớp **Caller** với mỗi hàm gọi một chuỗi thông tin. Cùng hàm gọi của lớp **Callme** qua mỗi lớp **Caller**.

```
//This program is not sychronize.
class Callme
{
        void  call(String msg)
        {
                System.out.print("[" + msg);
                try
                {
                        Thread.sleep(1000);
                }catch(InterruptedException e)
                        {
                                System.out.print("interrupted");
                        }
                System.out.print("]");
        }
}
class Caller implements Runnable
{
        String msg;
        Callme target;
        Thread t;
        public Caller(Callme targ,String s)
        {
                target=targ;
                msg=s;
                t=new Thread(this);
                t.start();
        }
        public void run()
        {
                target.call(msg);

        }

}
class Synch
{
        public static void main(String args[])
        {
                Callme target= new Callme();
                Caller ob1= new Caller(target,"Hello");
7               Caller ob2= new Caller(target,"Synchronized");
                Caller ob3= new Caller(target," World");
                //wait for threads to end
                try
                {
                        ob1.t.join();
                        ob2.t.join();
                        ob3.t.join();
                }catch(InterruptedException e)
                        {
```

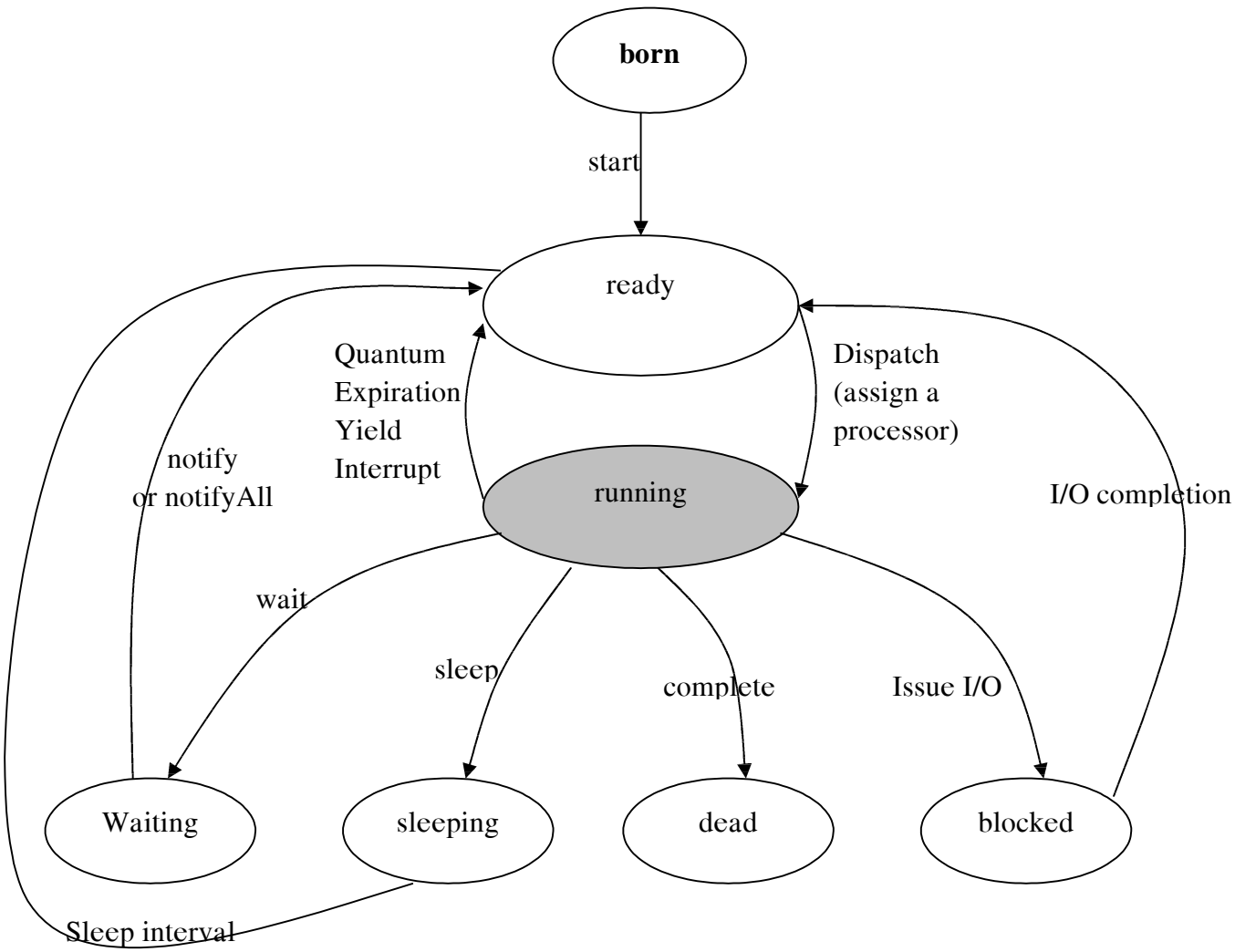
```
                    System.out.println("Interrupted");
                }
            }
        }
```

Kết quả chương trình sau:

Starting application E:\BTJava\Synch.class
[Hello[Synchronized[World]]]
Interactive Session Ended

Bằng cách gọi hàm sleep(), phương thức call() cho phép chuyển đổi thi hành qua lại giữa các thread. Kết quả trên xuất ra từ sự pha trộn ba chuỗi thông điệp. Trong chương trình này, không có gì ngưng cả ba tiến trình từ việc gọi cùng phương thức trên cùng một đối tượng tại một thời điểm. Điều này xem như cùng điều kiện tốc độ vì ba thread chạy riêng lẻ.

8. Các trạng thái của thread



Hình 16: *Các trạng thái của thread*

Thread có ba trạng thái chính:

- Trạng thái sẳn sàn (ready)
- Trạng thái thực thi(running)
- Trạng thái block (waiting, sleeping, deal, blocked)

Chu trình sống của một thread:

Trước tiên thread được sinh ra (born), đưa vào trạng thái sẳn sàng (ready), tiếp tục vào trạng thái phục vụ(running), trong thời gian phục vụ nếu thread hoàn tất thì thread đó bị hủy bỏ, hoặc chờ sự kiện(có thể là I/O) nó được đưa vào các trạng thái tương ứng nếu sự kiện đang chờ xảy ra nó tiếp tục đưa vào trạng thái sẳn sàng(ready) để tiếp tục xử lý cho hoàn tất.

Phần IV:
Yêu Cầu & Kiến Trúc Chương Trình

- ✓ Chương I: Yêu cầu chức năng & phi chức năng
- ✓ Chương II: Kiến trúc chương trình

Chương I: Yêu Cầu Chức Năng & Phi Chức Năng

1. Đối với Chat Client

Yêu cầu chức năng

- Chương trình cho phép người dùng lưu giữ thông tin về servername và port ngoài ra còn thông tin về nhóm server
- Cho phép thêm mới server
- Cho phép cập nhật lai thông tin server
- Cho phép hủy thông tin server
- Lưu giữ thông tin cấu hình hiện trạng của lần hoạt động trước ví dụ: thông tin về nickname, realname,email address, v.v...
- Cho phép user connect và disconnect trong các trường hợp sau

  Disconnect khi chưa connect xong ở mức TCP.

  Disconnect khi connect xong ở mức TCP nhưng chưa xong ở mức application (chưa thiết lập kết nối xong theo giao thức IRC).

  Disconnect khi đã xong ở mức application.
- Việc thiết lập kết nối và cắt đứt kết nối phải hoạt động theo cơ chế multithread để đảm bảo chương trình không bị block
- Các message được truyền, nhận, và xữ lý cũng phải được hoạt động theo cơ chế multithread nhằm bảo đảm chương trình hoạt động hiệu quả hơn.
- Chức năng tham gia vào một hay nhiều channel.
- Chức năng thoát khỏi channel.
- Về chức năng chat phải cho phép người sử dụng thực hiện chat với hai cách:

  Chat riêng với user.

  Chat trên channel.
- Chương trình phải cho phép người sử dụng chọn user để chat
- Chương trình cho phép người dùng Ban hay Kick user ra khỏi channel
- Chức năng thay đổi nick name hiện tại thành nick khác.
- Chức năng thiết lập cơ chế trả lời tự động (cơ chế away message).
- Chức Năng DCC(Direction Connection Chat) Chat.

Yêu cầu phi chức năng:

- Chức năng truy vấn user (Query User)
- Chức năng truy vấn Who is User.
- Chức năng send CTCP.
- Chức năng về giao diện: tùy chọn về màu sắc
- Chức năng chat thông qua proxy server

2. Đối với chat server

Yêu cầu chức năng

➢ Server phải phục vụ nhiều client trong cùng thời điểm, mỗi client được phục vụ liên tục.

➢ Chức năng Restart và ShutDown server

➢ Chức năng tùy chọn cấu hình cho server như chỉ số port, tên server, số lượng tối đa connection, số lượng tối đa channel được phép mở.

➢ Chức năng kiểm soát và quản lý user trên server, trên server có thể kiểm soát quá trình truyền nhận message của từng client.

Yêu cầu phi chức năng

➢ Chức năng theo dõi thông số hoạt động, tỉ lệ phần trăm tài nguyên mà server chiếm giữ

➢ Chức năng kết nối và chấp nhận kết nối với server khác, chuyển tiếp message đến server khác.

Chương II: Kiến trúc chương trình:

A. Chat Client

Chương trình gồm 2 nhóm lớp chính

1. Lớp giao diện

- frmAddNew

  Lớp frmAddNew tạo ra form cho phép user nhập vào server mới hoặt cập nhật lại thông tin server

- frmBanKick

  Lớp frmBanKick tạo ra form cho phép user có thể ban hay kick user nếu user có quyền

- frmChangeNick

  Lớp này cho phép user thay đổi nickname của mình.

- frmChannelStatus

  Lớp frmChannelStatus tạo ra khi user join thành công vào một channel trên form này gồm 3 control txtChannelStatus trạng thái channel, txtMessage, lstNickName đây là danh sách user hiện có trên channel danh sách này luôn được cập nhật và thay đổi khi có sự tham gia hay thoát khỏi của user.

- frmChannelStatusListenner

  Lắng nghe sự kiện của channelstatus

- frmChannelStatusKeylistener

  Lắng nghe sự kiện phím của channelstatus

- frmChatForm

  Khi thực hiện chat với riêng với từng user, mỗi user được chat sẽ có một frmChatForm

- frmConnect

  Lớp này sẽ hiển thị form connect to server trên form này có chức năng chọn server, Addserver Editserver, Deleteserver.v.v ...

- frmConnectListenEvent

  Lớp frmConnectListenEvent chịu trách nhiệm lắng nghe và xử lý sự kiện cho form connect.

- frmDCCChat

Lớp frmDCCChat hiển thị form DCCChat, form này cho phép user tạo ra connection mới kết nối trực tiếp vào user muốn chat với mình (Direction Connection Chat)

- frmGetListChannel

  lớp này cho phép user truy vấn đến server lấy về danh sách channel hiện có.

- frmJoinChannel

  Lớp frmJoinChannel cho phép user chọn channel để join tuy nhiên danh sách channel trên lớp này là những channel thông dụng đôi khi không có trên server trước khi join bạn nên dùng form getlistchannel.

- frmJoinChannelListener

  Lớp này chịu trách nhiệm lắng nghe và xữ lý sự kiện cho form join channel

- frmMainForm

  Lớp mainform là form chinh lớp này sẽ tồn tại khi chương trình tạo ra và duy trì suốt trong chương trình cho đến khi kết thúc. Trên lớp này lưu giữ các tham chiếu đến các đối tượng, có thể xem lớp này là lớp lưu giữ các biến toàn cục.

- frmMainFormListener

  frmMainFormListener lắng nghe và xữ lý sự kiện cho lớp frmMainForm

- frmMainFormMouseEvent

  frmMainFormMouseEvent lắng nghe và xữ lý sự kiện mouse cho lớp frmMainForm

- frmPartChannel

  lớp part channel dùng cho user thoát khỏi channel mà họ đã join

- frmSetAway

  Lớp frmSetAway dùng để thiết lập việc trả lời tự động

2.Lớp xữ lý

- threadIn

  Nhận message nhập vào phân phối message cho từng trường hợp trong quá trình thiết lập kết nối giữa client và server.

- threadOut

  Chịu trách nhiệm xuất message đến server.

- threadJoin
  Nhận message nhập vào và xữ lý cho từng trường hợp sau quá trình thiết lập kết nối thành công

- choiseItemListener
  Xữ lý cho sự kiện chọn combo box và listbox

- openSocket
  Lớp này chịu trách nhiệp mở kết nối ở cấp TCP liên kết đến server

- openServerSocket
  Lớp này mở ServerSocket lắng nghe kết nối cho dịch vụ DCCChat(Direction Connection Chat)

- DCCChatGetData
  Nhận message từ DCCChatServer

- DCCChatGetDataServer
  Nhận message từ DCCChatClient

- GetDataFromSocket
  Lớp này cho phép đọc dữ liệu từ luồng nhập và xuất dữ liệu từ luồng xuất lớp này phục vụ cho lớp threadIn và lớp threadOut.

- SettingFile
  Lớp SettingFile đọc và ghi file cấu hình hệ thống

- StatusButton
  Lớp này kế thừa từ Button Command
- channelFileProcess
  đọc và ghi file luu giữ danh sách channel

B. Chat Server

1. Lớp giao diện

- frmConfig
  Hiển thị form cấu hình cho server

- frmListUser
  Hiển thị danh sách user
- frmMainForm

form chính

2. Lớp xữ lý

- ChannelNameItem

  Lớp này tao ra đối tượng channelItem đối tượng này lưu giữ thông tin channel số lượng user hiện có, topic, tên channelv.v...

- GetDataFromSocket

  Lớp này nhận vào đối tượng socket và đọc và ghi dữ liệu lên luồng xuất nhập

- IRCServerSocket

  Lớp này chịu trách nhiệm tạo socket đợi và chấp nhận kết nối từ client, khi có client kết nối thành công (ở mức TCP) lớp này sẽ tạo ra một đối tượng UserConnectionItem truyền socket mà nó nhận được làm đối số.

- NickNameItem

  Một đối tượng UserConnectionItem sẽ là con của đối tượng NickNameItem, đối tượng này sẽ chứa thông số về cơ bản user như: nickname, hostname, username, realname.

- ProcessClientConnection

  Lớp này sẽ chịu trách nhiệm xử lý message khi message gởi đến server. Mọi xử lý trong lớp này do hàm DataArrive() xử lý

- StringProcess

  Lớp StringProcess là lớp bổ trợ cho việc xử lý chuỗi để phân tích message

- UserConnectionItem

  Đối tượng này được kế thừa từ NickNameItem lớp này chứa các thông số ChannelItem, đối tượng socket đang phục vụ cho nó.

- threadIn

  Lớp này có nhiệm vụ đọc dữ liệu nhập vào mà theo cơ chế multithread để chương trình không bị block

- threadOut

  Cũng tương tự như lớp threadIn lớp này có nhiệm vụ xuất message.

Để có thêm thông tin chi tiết cho các lớp đối tượng tham khảo danh sách các lớp chat client vàDanh Sách các lớp Server

Kết luận:

Kết quả đạt được:

1. Tìm hiểu được giao thức Internet Relay Chat.
2. Xây dựng chương trình chat client và chat server chuẩn.
3. Chat client có thể kết nối vào chat server chuẩn khác.
4. Chat server có thể làm việc được với chat client chuẩn khác.

Việc chưa làm được:

1. Chưa hổ trợ một số chức năng cho người quản trị.
2. Chat client chưa có khả năng chat thông qua proxy server.
3. Giao diện người dùng chưa thân thiện.

Phân công công việc:

Chung Diệu Tùng: - xây dựng các lớp giao tiếp với server theo giao thức chuẩn.
- xây dựng thành phần xử lý sự kiện.

Trần Thái Lộc: - xây dựng thành phần giao diện.
- xây dựng báo cáo.

Hướng phát triển đề tài:

Nếu có điều kiện phát triển tiếp tục đề tài em sẽ hoàn thiện thêm chương trình chat client bằng cách thêm vào các tính năng chat thông qua proxy server, multimedia chat và xây dựng chương trình multimedia chat server để giao tiếp với chat client phục vụ nhu cầu ngày càng cao của người sử dụng.

Tài liệu tham khảo:

☞ **Core Java volume I, II Fundamential**	**Cays.Horstmann, Gary Cornbll**
☞ **Giáo trình lý thuyết và bài tập Java**	**Trần Tiến Dũng**
☞ **Java How To Program**	**Deitel**
☞ **Java2**	**Nguyễn Tiên, Ngô Quốc Việt**
☞RFC 1459,	**J. Oikarinen, D. Reed – 05/1993**
☞RFC 2810,	**C. Kalt – 04/2000**
☞RFC 2811,	**C. Kalt – 04/2000**
☞RFC 2812,	**C. Kalt – 04/2000**
☞RFC 2813,	**C. Kalt – 04/2000**
☞ **TCP/IP MCSE Study Guide**	**GregP.Bulette**

Phụ Lục

Các giá trị trả về (REPLIES Value)

Đây là danh sách những giá trị trả về mà chúng được tạo ra khi nhận được message, mỗi giá trị trả về là một chuỗi, tương ứng với mỗi chuỗi là một giá trị số nguyên gồm 3 chữ số.

1. Error Replies

Số	Tên	Chuỗi trả về	Giải thích
401	ERR_NOSUCHNICK	<nickname>:No such nick / channel	Không có nickname trùng với nickname do user cung cấp
402	ERR_NOSUCHSERVER	<server name>:No such server	Đối số servername cung cấp không đúng
403	ERR_NOSUCHCHANNEL	<channel name>:No such channel	Đối số channel name không tồn tại
404	ERR_CANNOTSENDTOCHAN	<channel name> :Cannot send to channel	Không thể gởi đến channel
405	ERR_TOOMANYCHANNELS	<channel name>:You have join too many channel	Thông báo cho biết user đã tham gia quá số lượng cho phép
406	ERR_WASNOSUCHNICK	<nickname>:There was no such nickname	Thông báo không có nickname trong quá khứ
407	ERR_TOOMANYTARGETS	<target>:Duplicate recipients. No message \delivered	Thông báo cho client gởi PRIVMSG/NOTICE message đã nhận nhiều message giống nhau
409	ERR_NOORIGIN	:No origin specified	Ping, pong message không có đối số
410			
411	ERR_NORECIPIENT	:No recipient given (<command>)	
412	ERR_NOTEXTTOSEND	:No text to send	Không có text được gởi
413	ERR_NOTOPLEVEL	<mask>:No toplevel domain specified	Domain không có toplevel
414	ERR_WILDTOPLEVEL	<mask>:Wildcard in top level domain	Willcard trong toplevel domain
421	ERR_UNKNOWNCOMMAND	<command>:Unknown commad	Server thông báo cho client đã được đăng ký không hiểu lệnh nhận được
422	ERR_NOMOTD	:MOTD file is missing	
423	ERR_NOADMININFO	<server>: No	Không có thông tin người quản

		administrative info available.	lý
424	ERR_FILEERROR	:File error doing <file op> on <file>	Lỗi trong việc thao tác file
431	ERR_NONICKNAMEGIVEN	:No nickname given	Không có đối số nickname
432	ERR_ERRONEUSNICKNAME	<nick>:Erroneus nickname	Nick name không hợp lệ
433	ERR_NICKNAMEINUSE	<nick>:Nickname is already in use	Nick name đã dùng(có) xảy ra khi user thay đổi nickname
436	ERR_NICKCOLLOSION	<nick>:Nickname collision KILL	Server trả về thông báo khi có sự xung đột giữa 2 nickname
441	ERR_USERNOTINCHANNEL	<nick><channel>:They aren't on that channel	User không có trong channel
442	ERR_NOTONCHANNEL	<channel>:You are not on channel	Thông báo user không có trên channel được đưa ra
443	ERR_USERONCHANNEL	<user><channel>:is already on channel	User hiện đang có trên channel
444	ERR_NOLOGIN	<user>:user not logged in	Trả về sau khi nhận SUMMON message,thông báo user chưa login
445	ERR_SUMMONDISABLED	:SUMMON has been disabled	
446	ERR_USERSDISABLE	:USERS has been disabled	
451	ERR_NOTREGISTERED	:You have not registered	client chưa đăng nhập vào server
461	ERR_NEEDMOREPARAMS	<command>:Not enough parameters	Không đủ đối số cho command
462	ERR_ALREADYREGISTRED	:You may not reregister	Đã được đăng nhập
463	ERR_NOPERMFORHOST	:Your host isn't among the privileged	Server không cho phép host đăng nhập
464	ERR_PASSWDMISMATCH	:Password incorrect	Không đúng password
465	ERR_YOUREBANNEDCREEP	:You are banned from this server	Thông báo user đang ở trạng thái ban
467	ERR_KEYSET	<channel>:Channel key already set	
471	ERR_CHANNELISFULL	<channel>:Cannot join channel (+l)	Danh sách user trong channel đã đầy
472	ERR_UNKNOWNMODE	<char>:is unknown mode char to me	Không hiểu ký tự đại diện cho MODE
473	ERR_INVITEONLYCHAN	<channel>:Cannot join channel (+i)	Channel đang ở trạng thái +I invite only nên user không thể

			JOIN
474	ERR_BANNEDFROMCHAN	<channel>:Cannot join channel (+b)	User bị ban khỏi channel nên không thể JOIN
475	ERR_BADCHANNELKEY	<channel>:Cannot join channel (+k)	
481	ERR_NOPRIVILEGES	:Permission Denied- you're not an IRC operator	User không có quyền thực thi vì không có quyền channel opreation
482	ERR_CHANOPRIVSNEEDED	<channel>:you're not channel operator	Không được lấy quyền operation vì đã có user làm channel operation
483	ERR_CANTKILLSERVER	:You can't kill a server	client không thể "KILL" server
491	ERR_NOOPERHOST	:No O-line for your host	Server thông báo không thể cho "host" này làm "chanop"
501	ERR_UMODEUNKNOWNFLAG	:Unknown MODE flag	Không hiểu MODE flag
502	ERR_USERSDONTMATCH	:Can change mode flag sent was not recognized	Thông báo khi có user thay đổi MODE của user khác mà nó không được quyền

2. Command Responses

Số	Tên	Chuỗi trả về	Giải thích
200	PRL_TRACELINK	Link<version & debug level> <destination>\<next server>	
201	PRL_TRACECONNECTING	Try.<class><server>	
202	PRL_TRACEHANDSHAKE	H.S. <class><server>	
203	PRL_TRACEUNKHOWN	???? <class>[<client IP address in dot form>]	
204	PRL_TRACEOPERATOR	Oper <class><nick>	
205	PRL_TRACEUSER	User <class><nick>	
206	PRL_TRACESERVER	Serv<class><int>S <int>C <server>\<nick!user| *!* > @<host | server>	
208	PRL_TRACENEWTYPE	<newtype> 0 <client name>	
211	PRL_STATSLINKINFO	<linkname><sendq><sent messages>\<sent byte> <receive message> \ <receivebyte> <timeopen>	
212	PRL_STATSCOMMANDS	<command><count>	

213	PRL_STATSCLINE	C <host> * <name><port> <class>	
214	PRL_STATSNLINE	N <host> * <name><port> <class>	
215	PRL_STATSILINE	I <host> * <name><port> <class>	
216	PRL_STATSKLINE	K <host> * <username> <port> <class>	
218	PRL_STATSYLINE	C <class><pingfrequency> <connect\ frequency> <max sendq>	
219	PRL_ENDOSTATS	<stats letter>:End of /STATS report	
221	PRL_UMODEIS	<user mode string>	
241	PRL_STATSLLINE	L<host mask>* <servername><maxdepth>	
242	PRL_STATSUPTIME	:Server Up %d day %d :%02d:%02d	
243	PRL_STATSOLINE	O <hostmask> * <name>	
244	PRL_STATSHLINE	H <hostmask> * <servername>	
251	PRL_LUSERCLIENT	:there are <integer>users and <integer>invisible on <integer> server	
252	PRL_LUSEROP	<integer> :operator(s) online	
253	PRL_LUSERUNKNOWN	<integer> :Unknown connection(s)	
254	PRL_LUSERCHANNEL	<integer> :Channels formed	
255	PRL_LUSERME	:I have <integer> client and <integer> / server	
256	PRL_ADMINME	<server> :Administrative info	
257	PRL_ADMINLOC1	:<admin info>	
258	PRL_ADMINLOC2	:<admin info>	
259	PRL_ADMINMAIL	:<admin info>	
261	PRL_TRACELOG	File <logfile><debug level>	
300	PRL_NONE		
301	PRL_AWAY	<nick>:<away message>	
302	PRL_USERHOST	:[<reply>	

		{<space><reply>}]	
303	PRL_ISON	:[<nick> {<space><nick>}]	
305	PRL_UNAWAY	:You are no longer marked as being away	
306	PRL_NOWAWAY	:You have been marked as being away	
311	PRL_WHOISUSER	<nick><user><host> * :<realname>	
312	PRL_WHOISSERVER	<nick><server>:<server info>	
313	PRL_WHOISOPERATOR	<nick>:is an IRC operator	
314	PRL_WHOWASUSER	<nick><>user<host> * :<real name>	
315	RPL_ENDOFWHO	<name> :End of /Who list	
317	PRL_WHOISIDLE	<nick><integer>:seconds idle	
318	PRL_ENDOFWHOIS	<nick>:End of /WHOIS list	
319	PRL_WHOISCHANNELS	<nick>:{[@|+]<channel> <space>}	
321	PRL_LISTSTART	Channel :Users Name	
322	PRL_LIST	<channel><#visible> : <topic>	
323	PRL_LISTEND	:End of /LIST	
324	PRL_CHANNELMODEIS	<channel><mode> <more params>	
331	RPL_NOTOPIC	<channel> :No topic is set	
332	RPL_TOPIC	<channel> :<topic>	
341	RPL_INVITING	<channel><nick>	
342	RPL_SUMMONING	<user>:Summoning user to IRC	
351	RPL_VERSION	<version>.<debuglevel> <server>	
352	RPL_WHOREPLY	<channel><user><host> <server><nick> \ <H | G> [*][@|+]:<hopcount> <real name>	
353	RPL_NAMREPLY	<channel>:[[@|+] <nick> [[@|+] <nick>	
364	RPL_LINKS	<mask > <server> :<hopcount><server info>	
365	RPL_ENDOFLINKS	<mask> :End of /LINK list	

366	RPL_ENDOFNAMES	<channel> :End of /NAME List	
367	RPL_BANLIST	<channel><banid>	
368	RPL_ENDOBANLIST	<channel>:End of channel ban list	
369	PRL_ENDOFWHOWAS	<nick>:End of WHOWAS	
371	RPL_INFO	:<string>	
372	RPL_MOTD	- <text>	
374	RPL_ENDOINFO	:End of /INFO list	
375	RPL_MOTDSTART	- <server> Message of the day -	
376	RPL_ENDOFMOTD	:End of/MOTD command	
381	RPL_YOUROPER	:You are now an IRC operator	
382	RPL_REHASHING	<config file> :rehashing	
391	RPL_TIME	<server> :<string showing server's local time >	
392	RPL_USERSSTART	:UserID Terminal Host	
393	RPL_USERS	:%-8s %-9s %-8s	
394	RPL_ENDOUSERS	:End of users	
395	RPL_NOUSERS	:No body logged in	

Các giá trị số dành riêng(Reserved Numerics)

số	Giá trị chuỗi	Giải Thích
209	RPL_TRACECLASS	
231	RPL_SERVICEINFO	
233	RPL_SERVICE	
235	RPL_SERVLISTEND	
316	RPL_WHOISCHANOP	
362	RPL_CLOSING	
373	RPL_INFOSTART	
466	ERR_YOUWILLBEBANNED	
492	ERR_NOSERVICEHOST	
217	RPL_STATQLINE	
232	RPL_ENDOFSERVICES	
234	RPL_SERVLIST	
361	RPL_KILLDONE	
363	RPL_CLOSEEND	
384	RPL_MYPORTIS	

476	ERR_BADCHANMASK	