Sunanth Sakthivel
CS594

Internet Relay Chat Protocol (RFC)

CONTENTS:

1. Introduction

The proposed protocol implementation is a Internet Relay Chat (IRC) protocol. This specific implementation uses a “relay” system where the central server essentially relays messages between client users that are connected with the central system. In other words, users are able to join private rooms and have their messages broadcasted to other end user clients who are also connected to the central server.

2. Assumptions

Data transport between client and server is purely through the internet protocol (IP) and Transmission control protocol (TCP) as the desired transport protocol. A persistent TCP connection is maintained between the client and server to allow for the delivery of messages between one another. The server and client are independent from each other and therefore either of the two can disconnect independently, the protocol is not always responsible for unexpected or abrupt disconnection from either parties. Lastly, the protocol will not instill a limitation on how many clients are allowed to join a server at any given time--ultimately, the only real limitation is how much the server can actually support.

3. IRC Layout

There will be a central server where clients can connect to. The clients that are connected will depend on the server to carry out specific functionalities such as messaging and joining channels. When a client connects he/she is assigned a unique nickname of their choosing, if it already exists then the unique port number is appended at the end of the name. Clients can join multiple channels and can furthermore chat with other clients who are also in the same channels. Channels will be limited to a maximum of 10 clients and will persist on the server even when no one is within a channel. Messaging is straightforward in the sense that messages are sent to the server which will then determine which clients it must route the information to. In the case of private messaging, the server will send the information to a single client. Group messaging in a channel will comprise of the server sending the messaging out to several clients within the channel.

4. Client/server commands

The server is responsible for receiving and parsing the information sent from the clients and will thus respond appropriately. Depending on the information passed in, the server may carry out the desired action requested by the client or it may reply with an error.

4.1 General structure

Client side:
<command> <arguments>

Server side:
<command> <arguments>

4.2 Message details

Command: refers to the actual command issued by the client or server and may or may not take in argument parameters. Commands are read in by standard input (sys.stdin) and processed appropriately depending on the command and arguments issued.
Arguments: not all commands require arguments, if a command requires an argument and no argument is given then server replies with error.

4.3 Server Commands

This are commands issued by the server itself and will be processed by the server. The commands provide basic functionality for the server such as graceful disconnection, removal of client and display of all current users/channels.

4.3.1 KICK

The server can use the KICK command when it wants to remove a specific client from the server. This command provides a graceful way of removing the client from the server without impacting the book keeping of other resources. The server will simply remove the client assigned with the specific username if it exists.

Server:
KICK <username>
Server response:
“Kicked from server”
“<user> does not exist in server”

The username argument is the username that is assigned to a specific client. If the username does not exists then server responds to itself with an error stating that the user does not exist.

4.3.2 DC

The server can use the DC command when it wants to gracefully disconnect. This graceful exit will inform all the clients that are connected that the server is shutting down. There are no error responses with this command and no arguments are given.

Server:
DC <no args>
Server response:
“Server shutting down”

Command DC is only needed, no arguments are given.

4.3.3 INFOALL

The server can use the INFOALL command to process all the stored data structures holding information regarding all the clients and channels that they are in. This information is displayed out in standard output. There are no error responses with this command and no arguments are given.

Server:
INFOALL <noargs>
Server response:
<display list of users in server>
<display list of channels in server>
<display channel with corresponding users>

Command INFOALL is only needed, no arguments are given.

4.4 Client Commands

These are commands issued by the client and are then processed by the server which will determine whether or not to carry out the desired request. Moreover, the client is dependent on the server to carry out all the IRC features. Commands include features such as messaging and joining/leaving channels.

4.4.1 EXIT

The client can use the EXIT command to tell the server that it is disconnecting its TCP connection with the server. The server will then end the connection with the client. This provides a graceful way for the client to disconnect from the server. The server will not respond to this command with any errors and no arguments are needed.

Client:
EXIT <noargs>
Server response:
“Client has left channel”

Command EXIT is only needed, no arguments are given. The server will respond by informing all other clients that are within the same channels that the user has left the channel.

4.4.2 REG

The client implicitly uses the REG command when it first connects to the server to associate its unique ephemeral port number with a username. The client can also use this command after connected to change his/her username. The server is responsible for storing the client’s username in a data structure for future bookkeeping.

Client:
REG <username>
Server response:
If username exists: “username already exists, signed in as <usernameport>”
Otherwise: <add name>

The username argument is used for the desired username. If the username already exists then the server responds by informing the client and appending the client’s unique ephemeral port number to the end of the username desired--this guarantees that the username will always be unique.

4.4.3 JOIN

The client can use the JOIN command to join a desired channel. Clients who are in a channel can message all other clients who are in the same channel. The server processes the JOIN request by checking if the channel exists, if the user is already in the channel, and if the maximum number of clients already occupy the channel. If all conditions are met, then the client is added to the channel and this information is recorded by the server data structure mapping. If the user decides to join a channel that doesn't exist then a new channel is created under that name.

Client:
JOIN <channel>
Server response:
If not enough args: “Error, try: JOIN <Channel>”
If new channel: “New channel <channel> made and joined by <user>”
If channel exists: “<user> has joined channel”
If already in channel: “Already in channel”
If max users in channel: “channel has max users”

The channel argument is the channel the client wishes to join. Server errors: if the user doesn't provide the channel argument then server responds with needing more args. If client attempts to join a channel he/she is already in then server informs client that he/she is already in that channel. Lastly, if the client attempts to join a channel that already has max users then the server refuses to add the client to the channel.

4.4.4 LIST

The client can use the LIST command to display all the channels that exist in the server. The server will respond by processing the available channels that exist in the server and forwarding that information to the client.

Client:
LIST <noargs>
Server response:
If no channels: “no channels exist on server”
Otherwise: “List of channels <list>”

This command takes no arguments. The server responds with a list of channels available, if no channels exist on the server, then this will be informed to the client.

4.4.5 MEM

The client can use the MEM command to display all the members within a desired channel. The client need not be a part of the channel to display the members of a specific channel. The server responds by processing the list of users within the desired channel and forwarding this information to the client. The server will also check to make sure there are enough arguments given and if the channel provided even exists.

Client:
MEM <channel>

Server response:
If not enough args: “Error, try: MEM <channel>”
If channel doesn’t exist: “<channel> does not exist”
If channel contains no users: “no users in <channel>”
If channel exists with users: “users in <channel> <list>”

The channel argument is the desired channel the client wants to query for users. Server errors: if the user doesn’t provide the channel argument then server responds with needing more args. If the client tries to check for members of a channel that doesn’t exist then the server will respond accordingly.

4.4.6 LEAVE

When the client uses the LEAVE command then the server responds by essentially removing the client from the chosen channel argument. The server will do this by updating the data structures that store the user to channel mapping. Once the client is removed from the channel, then the server informs all other clients that are a part of this channel that the user has left the channel. Additionally, the server will check if the channel even exists and if the client exists in the channel to begin with. If no argument is provided then the server attempts to remove the client from all channels he/se resides in.

Client:
LEAVE <noargs>
LEAVE <channel>

Server response:
If no args given: “Leaving all channels”
If channel does not exist (arg given): “This channel does not exist”
If not within the channel (arg given): “not currently in <channel>”
Otherwise (arg given): “Client has left channel”

The channel argument is the channel the client wishes to leave. Server errors: if the user doesn’t provide the channel argument then server responds with needing more args. If the channel doesn’t exist on the server then the client is informed. If the client attempts to leave a channel he/she is not even in, then the server responds without making any updates.

4.4.7 MESS

When the client uses the MESS command it allows the client to send messages to all other clients within the chosen channel. The server will check to ensure that the user is within the channel he/she is trying to message, if the correct number of argument parameters are given, and if the channel even exists. If the conditions are met, the server will send the information to all the clients who exist on the chosen channel.

Client:
MESS <channel> <message>
Server response:
If not enough args: “Error, try: Mess <channel><message>”
If channel does not exist: “This channel does not exist”
If not in channel: “Not currently in <channel>”
Otherwise: <send message>

The channel argument is the desired channel the client wishes to send the message to. The message argument is the message the user wants to send. Server errors: If the client does not provide the correct number of arguments then the server responds with needing more args. If the channel doesn’t exist on the server then the client is informed. Lastly, if the client attempts to send a message to a channel that he/she is not in the server will respond by not forwarding the message to the clients of the channel and inform the client.

4.4.8 WHISPER

When the client uses the WHISPER command it allows them to private message another client that is also connected to the same server. The server will check to ensure that the username exists, if the correct number of arguments are given, or if the user attempts to private message self. If the conditions are met then the server will appropriately forward the message to the desired client privately.

Client:
WHISPER <username> <message>
Server response:
If not enough args: “Error, try: WHISPER <username> <message>”
If whispering self: “Unable to private message self”
If user does not exist: “User <username> does not exist”
Otherwise: <send message>

The username argument is the user the client wishes to private message. The message argument is the message the client wants to send. Server errors: If the client does not provide the correct number of arguments then the server responds with needing more args. If the username doesn’t exist on the server then the client is informed. Lastly, if the client tries to private message him/herself then the server notifies the client that this isn’t possible.

4.4.9 GET

When the client uses the GET command it allows the client to receive the desired file from the server if it exists. The server will check if the file exists and if the correct number of parameters are given when issuing this command. If conditions are met, the server will open the file and write out the contents to the client where the client will write this information into the transferred file.

Client:
GET <filename.extension>
Server response:
If not enough args: “Error, try: Get <filename>”
If file does not exist: “File does not exist”
Otherwise: <send file to client>

The filename argument is the name of the file the client wishes to receive from the server. Server errors: If the client does not provide the correct number of arguments then the server responds with needing more args. If the server does not have the chosen file then the server notifies the client that the file doesn’t exist.

5. Security and error handling

Errors are appropriately handled by the server by responding to the client appropriately. Errors may lead to the server sending back a message to the client indicating that an error was made or may just result in the disconnection of the server from the client when deemed necessary (such as when the connected client socket no longer responds).