PROJECT 1 README

CS457 Francisco R. Ortega

Jonathan Cowles

Michael Whitehill

VIDEO LINK:

Code Review:

https://youtu.be/EscnABKYebs

Code Usage/Demonstration:

https://youtu.be/dOsaXjM0XwY

CHAT SERVER USAGE:

- To start a new chat server a config file must be written and provided as an argument to the server, example server config file setup is below: **(Bold options are required)**

-s (server name)- can be left empty

-p (port #)

-b (desired banned user(s) text directory file path)

-u (desired user(s) list text directory file path)

-c (desired channel(s) list text directory file path)

-r (server rules) – can be left empty

-v (server version) – can be left empty

CHAT CLIENT USAGE:

- To start a client a config file may be written and the file path passed as an argument or manual arguments can be passed, example client config file setup is below: **(Bold options are required)**

-u (username) – if left empty username will not be specified and no private messages will be received by this user

-h (hostname) – should be "localhost" if running both server and client locally on the same machine

-p (port #) – should be set to the same port# as the server

-t (file path to test file)

-L (desired client log output text directory file path)

-P (client password) – password is '@' if no password is desired. Password must be provided if username is not empty and 'empty' password value of '@' cannot be used if non empty username is provided

-a (privilege level) – can be left empty and will be defaulted to 'user'. Valid options are 'user', 'channelop', 'sysop', 'admin'

Both the Server and the Client are run through **main.cpp – SERVER MUST BE STARTED BEFORE CLIENT(S)**

To run the server side the arguments are:

- "[filepath to server configuration file], srv"

To run the client side the 2 options for arguments are:

- Manual arguments in the same form as config file "-u 'username' … , clt"
- "[filepath to client config file], clt"
-

CLIENT USAGE PARAMETERS:

User(s) will be prompted for an OP code and then will be further prompted for any input that is required for that OP code. OP codes like JOIN or PRIVMSG can be entered in upper or lowercase. No other formatting is accepted or user will be prompted to re-enter OP code.

It is advised to make a directory of logs with subdirectories 'client' and 'server' to distinguish between server and client log files.

Below are all of the supported 'OP codes' that are recognized by the server from the client. Invalid codes will result in error message and prompt user to input a recognized OP code.

The Description cell is formatted with (RFC description) (**our implementation**)

	COMMAND	DESCRIPTION
NOT Supported	**CONNECT**	Instructs the server <remote server> (or the current server, if <remote server> is omitted) to connect to <target server> on port <port>.[3][4] This command should only be available to IRC operators **We implemented this through a JSON command INITIAL_SETTINGS that sends username, password & level to the server as a default first message send in client.cpp – username field is blank for unregistered user and password is '@'**

DONE	**DIE**	Instructs the server to shut down
DONE	**HELP**	Requests the server to display the help file. This command is not formally defined in an RFC, but is in use by most major IRC daemons. **Works per RFC, returns client JSON valid usage**
DONE	**INFO**	Returns information about the <target> server, or the current server if <target> is omitted. Information returned includes the server's version, when it was compiled, the patch level, when it was started, and any other information which may be considered to be relevant **INFO returns server name, start time and current time**
DONE	**ISON**	Queries the server to see if the clients in the space-separated list <nicknames> are currently on the network.[10] The server returns only the nicknames that are on the network in a space-separated list. If none of the clients are on the network the server returns an empty list. **Works per RFC, returns true if user is currently on the server and false otherwise. Works for one specified username at a time.**
DONE	**LIST**	Lists all channels on the server. If the comma-separated list <channels> is given, it will return the channel topics. If <server> is given, the command will be forwarded to <server> for evaluation. **Works per RFC, Lists all current registered channels on server**
NOT Supported	**MODE**	The MODE command is dual-purpose. It can be used to set both user and channel modes. **We chose to not implement modes and instead do everything through user levels: user,channelop,sysop,admin**
DONE	**PING**	Tests the presence of a connection. A PING message results in a PONG reply. If <server2> is specified, the message gets passed on to it. **Works per RFC, returns PONG if server connection is valid**
DONE	**PONG**	This command is a reply to the PING command and works in much the same way. **Works per RFC, returns PONG message to sending client**
NOT Supported	**RESTART**	Restarts a server. It may only be sent by IRC operator. **We did not implement a restart command for the server on the first section of this project**
DONE	**RULES**	Requests the server rules. **Works per RFC, returns string representation of current server rules as implemented in server config file**
DONE	**SILENCE**	Adds or removes a host mask to a server-side ignore list that prevents matching users from sending the client messages. More than one mask may be specified in a space-separated list, each item prefixed with a "+" or "-" to designate whether it is being added or removed. Sending the command with no parameters returns the entries in the client's ignore list. **Works per RFC, checks for correct privileges in order to silence the provided username. Once a user has been silenced, they cannot send PRIVMSG or MSG to any channel until they have been unsilenced. We treated SILENCE as a ban on a user and this user is added to the server ban list log. To unban/unsilence a user the silencing admin will use SILENCE op code with the same username again**
DONE	**TIME**	Returns the local time on the current server, or <server> if specified. **Works per RFC, returns current time from server**
DONE	**VERSION**	Returns the version of <server>, or the current server if omitted. **Works per RFC, returns server version as specified in the server configuration file**
DONE	**AWAY**	Syntax: AWAY [<message>] Provides the server with a message to automatically send in reply to a PRIVMSG directed at the user, but not to a channel they are on.[2] If <message> is omitted, the away status is removed. **Client user is prompted to enter AWAY message. If message 'here' is provided then AWAY status is cleared.**

DONE	**INVITE**	Invites <nickname> to the channel <channel>.[9] <channel> does not have to exist, but if it does, only members of the channel are allowed to invite other clients. If the channel mode i is set, only channel operators may invite other clients. **We have not implemented -i or -t flags for channels or users. Everything uses privilege levels. User must be invited to a channel that already exists.**
DONE	**JOIN**	Makes the client join the channels in the comma-separated list <channels>, specifying the passwords, if needed, in the comma-separated list <keys>.[11] If the channel(s) do not exist then they will be created **Works per RFC, channel will be created if it does not already exist. Only one channel to be joined can be specified at a time. Use multiple JOIN commands to join multiple different channels**
DONE	**KICK**	Forcibly removes <client> from <channel>. **Works per RFC, only users of privilege levels 'channelop', 'sysop' or 'admin' can kick users from a specific channel**
DONE	**KILL**	Forcibly removes <client> from the network. **Works per RFC, user must currently exist on the server and only users of privilege level 'sysop' or 'admin' can kick users from the server**
DONE	**KNOCK**	Sends a NOTICE to an invitation-only <channel> with an optional <message>, requesting an invite. **Works per RFC, channel must exist and then user can be invited to that specific channel. We did not implement an option to add an optional message**
DONE	**NICK**	Allows a client to change their IRC nickname. Hopcount is for use between servers to specify how far away a nickname is from its home server. **Works per RFC, for this project we have treated nicknames and usernames as the same thing so using OP code NICK calls SETNAME**
DONE	**NOTICE**	This command works similarly to PRIVMSG, except automatic replies must never be sent in reply to NOTICE messages. **Works per RFC, user will receive NOTICE message but automatic reply will not be sent to sending user if status for receiving user is currently AWAY**
DONE	**PART**	Causes a user to leave the channels in the comma-separated list <channels>. **Works per RFC, any user that is currently active on a particular channel can use PART to leave the specified channel**
DONE	**OPER**	Authenticates a user as an IRC operator on that server/network. **Works per RFC, returns true for privilege level of 'sysop' or 'admin' and returns current channels that an operator is registered as joined to if user is level 'channelop'**
DONE	**PASS**	Sets a connection password.[28] This command must be sent before the NICK/USER registration combination. **When a user connects their password is set from config details by default, PASS allows for password to be changed for registered users only. Unregistered users must set a name with SETNAME command first**
DONE	**PRIVMSG**	Sends <message> to <msgtarget>, which is usually a user or channel. **Works per RFC, sends a PRIVMSG to a user that is currently registered with a username & password on the server – Users that are currently SILENCE(banned) cannot send PRIVMSGS**
DONE	**QUIT**	Disconnects the user from the server. **Works per RFC, Client will be disconnected from the server**
DONE	**SETNAME**	Allows a client to change the "real name" specified when registering a connection. **Works per RFC, allows users to set the name they want to be registered under on the server**
DONE	**TOPIC**	Allows the client to query or set the channel topic on <channel>.[44] If <topic> is given, it sets the channel topic to <topic>. If channel mode +t is set, only a channel operator may set the topic. **Works per RFC, channel mode(s) -+t are not implemented on this project. Only users with privilege level 'channelop' may set the TOPIC of a specific channel**
NOT Supported	**USER**	This command is used at the beginning of a connection to specify the username, hostname, real name and initial user modes of the connecting client.[46][47] <realname> may contain spaces, and

		thus must be prefixed with a colon. **USER command is implemented through SET_INITIAL command that is called by default when a user starts a client connection**
NOT Supported	**USERHOST**	Returns a list of information about the nicknames specified. **This command is implemented through WHO command**
NOT Supported	**USERIP**	Requests the direct IP address of the user with the specified nickname. This command is often used to obtain the IP of an abusive user to more effectively perform a ban. It is unclear what, if any, privileges are required to execute this command on a server. **This command is not implemented as there is no defined way to do this in C++ that works for all OS's**
DONE	**USERS**	Returns a list of users and information about those users in a format similar to the UNIX commands who, rusers and finger. **Works per RFC, returns name and level of each user currently active on the server**
DONE	**WALLOPS**	Sends <message> to all operators connected to the server (RFC 1459), or all users with user mode 'w' set **Works per RFC, we did not implement the 'w' mode so a message will be broadcasted to all current server users if it is sent by a user of privilege level 'sysop' or 'admin'**
DONE	**WHO**	Returns a list of users who match <name>.[53] If the flag "o" is given, the server will only return information about IRC operators. **Works per RFC, pattern matches all users that have part or all of the search string provided in their name**
NOT Supported	**WHOIS**	Returns information about the comma-separated list of nicknames masks <nicknames>.[54] If <server> is given, the command is forwarded to it for processing. **This is implemented for a singly user by calling the WHO command and specifying all or part of a username**
CUSTOM	**LOCK**	**LOCK allows channelop to lock a channel on the server**
CUSTOM	**UNLOCK**	**UNLOCK allows channelop to unlock a previously locked channel on the server**
CUSTOM	**MSG**	**MSG is the OP code command for sending a message to all users of a specified channel that the sender is currently a member of**