CS 3337
Homework 2 IRC Project

In this project, we will produce an IRC client and server according to the below standards. This project will be turned in over several deliverables.

We will produce an IRC client and server using several technologies that we will be learning about through the semester. These client and server programs will communicate with each other to broadcast messages to several clients according to channels the client has subscribed to. The requirements and protocol are influenced by RFC 1459 - Internet Relay Chat Protocol.

Requirements:

Client

- Connects to a server, transmitting nick name, user name, host name, server name, and real name
- Accepts messages to send to server including commands
- Displays messages from server including responses to commands or new messages from other users
- Receives messages from server to update metadata and capabilities (not necessarily shown immediately to user)
- Allows for joining multiple channels
- Configurable by user to mute channels, notify on all messages, notify on mentions (@nick)
- Implemented in command line using ncurses to provide an advanced graphical interface within a command line environment
- Provides support for configuration settings to be specified in a file and on the command line
- Provides support for logging/debugging

Server

- Allows multiple client connections
- Performs login actions including nick collision checking and audit logging
- Maintains blacklist/whitelist of nicks
- Maintains list of IRC operators (server admins)
- Maintains list of Channel operators (channel mods/admins)
- Maintains list of currently connected nicks
- Maintains list of currently active channels
- Receives messages from clients for channel
- Broadcasts messages to all connected clients on channel
  - Includes updates to channel metadata
- Supports sending private message only to intended recipient
  - Including responses to commands
- Receives and processes commands from client
- Receives and processes commands from server command line

Support Commands

- PASS <password>
  The PASS command is used to set a 'connection password'. The password can and must be set before any attempt to register the connection is made. Currently this requires that clients send a PASS command before sending the NICK/USER combination. It is possible to send multiple PASS commands before registering but only the last one sent is used for verification and it may not be changed once registered.

  Numeric Replies:
  ERR_NEEDMOREPARAMS
  ERR_ALREADYREGISTRED

  Example:
  PASS secretpasswordhere

- NICK <nickname>
  NICK message is used to give user a nickname or change the previous one.

  Numeric Replies:
  ERR_NONICKNAMEGIVEN
  ERR_ERRONEUSNICKNAME
  ERR_NICKNAMEINUSE

  Example:
  NICK Wiz ; Introducing new nick "Wiz".
  :WiZ NICK Kilroy ; WiZ changed his nickname to Kilroy.

- USER <username> <hostname> <servername> <realname>
  The USER message is used at the beginning of connection to specify the username, hostname, servername and realname of a new user.

  Note that hostname and servername are normally ignored by the IRC server when the USER command comes from a directly connected client (for security reasons).

  It must be noted that realname parameter must be the last parameter, because it may contain space characters and must be prefixed with a colon (':') to make sure this is recognised as such.

  Numeric Replies:
  ERR_NEEDMOREPARAMS
  ERR_ALREADYREGISTRED

  Example:
  USER guest tolmoon tolsun :Ronnie Reagan
  ; User registering themselves with a username of "guest" and real name "Ronnie Reagan".

- OPER <user> <password>
  OPER message is used by a normal user to obtain operator privileges. The combination of <user> and <password> are required to gain Operator privileges.

  If the client sending the OPER command supplies the correct password for the given user, the server then informs the rest of the network of the new operator by issuing a "MODE +o" for the clients nickname.

Numeric Replies:
ERR_NEEDMOREPARAMS
RPL_YOUREOPER
ERR_NOOPERHOST
ERR_PASSWDMISMATCH

Example:
OPER foo bar ; Attempt to register as an operator using a username of "foo" and "bar" as the password.

- QUIT [<Quit message>]
  A client session is ended with a quit message. The server must close the connection to a client which sends a QUIT message. If a "Quit Message" is given, this will be sent instead of the default message, the nickname.

  If, for some other reason, a client connection is closed without the client issuing a QUIT command (e.g. client dies and EOF occurs on socket), the server is required to fill in the quit message with some sort of message reflecting the nature of the event which caused it to happen.

  Numeric Replies:
  None.

  Examples:
  QUIT :Gone to have lunch ; Preferred message format.

- JOIN <channel>{,<channel>} [<key>{,<key>}]
  The JOIN command is used by client to start listening a specific channel. Whether or not a client is allowed to join a channel is checked only by the server the client is connected to. The conditions which affect this are as follows:

  1. the user must be invited if the channel is invite-only;
  2. the user's nick/username/hostname must not match any active bans;
  3. the correct key (password) must be given if it is set.

  These are discussed in more detail under the MODE command (see section 4.2.3 of RFC1459 for more details).

  Once a user has joined a channel, they receive notice about all commands their server receives which affect the channel. This includes MODE, KICK, PART, QUIT and of course PRIVMSG/NOTICE.

  If a JOIN is successful, the user is then sent the channel's topic (using RPL_TOPIC) and the list of users who are on the channel (using RPL_NAMREPLY), which must include the user joining.

  Numeric Replies:
  ERR_NEEDMOREPARAMS
  ERR_BANNEDFROMCHAN
  ERR_INVITEONLYCHAN
  ERR_BADCHANNELKEY
  ERR_CHANNELISFULL
  ERR_BADCHANMASK
  ERR_NOSUCHCHANNEL
  ERR_TOOMANYCHANNELS
  RPL_TOPIC

  Examples:
  JOIN #foobar ; join channel #foobar.

JOIN &foo fubar ; join channel &foo using key "fubar".
JOIN #foo,&bar fubar ; join channel #foo using key "fubar" and &bar using no key.
JOIN #foo,#bar fubar,foobar ; join channel #foo using key "fubar", and channel #bar using key "foobar".
JOIN #foo,#bar ; join channels #foo and #bar.

- PART <channel>{,<channel>}
  The PART message causes the client sending the message to be removed from the list of active users for all given channels listed in the parameter string.

  Numeric Replies:
  ERR_NEEDMOREPARAMS
  ERR_NOSUCHCHANNEL
  ERR_NOTONCHANNEL

  Examples:
  PART #twilight_zone ; leave channel "#twilight_zone"
  PART #oz-ops,&group5 ; leave both channels "&group5" and "#oz-ops".

- MODE <channel> {[+|-]|o|p|s|i|t|n|b|v} [<limit>] [<user>] [<ban mask>]
  The MODE command is provided so that channel operators may change the characteristics of 'their' channel. It is also required that servers be able to change channel modes so that channel operators may be created.

  The various modes available for channels are as follows:

  o - give/take channel operator privileges;
  p - private channel flag;
  s - secret channel flag;
  i - invite-only channel flag;
  t - topic settable by channel operator only flag;
  n - no messages to channel from clients on the outside;
  m - moderated channel;
  l - set the user limit to channel;
  b - set a ban mask to keep users out;
  v - give/take the ability to speak on a moderated channel;
  k - set a channel key (password).

  When using the 'o' and 'b' options, a restriction on a total of three per mode command has been imposed.

  Numeric Replies:
  ERR_NEEDMOREPARAMS
  RPL_CHANNELMODEIS
  ERR_CHANOPRIVSNEEDED
  ERR_NOSUCHNICK
  ERR_NOTONCHANNEL
  ERR_KEYSET
  RPL_BANLIST
  RPL_ENDOFBANLIST
  ERR_UNKNOWNMODE
  ERR_NOSUCHCHANNEL

Examples:
Use of Channel Modes:

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 channel #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.

- MODE <nickname> {[+|-]|i|w|s|o}
  The user MODEs are typically changes which affect either how the client is seen by others or what 'extra' messages the client is sent. A user MODE command may only be accepted if both the sender of the message and the nickname given as a parameter are both the same.

  The available modes are as follows:

  i - marks a users as invisible;
  s - marks a user for receipt of server notices;
  w - user receives wallops;
  o - operator flag.

  If a user attempts to make themselves an operator using the "+o" flag, the attempt should be ignored. There is no restriction, however, on anyone 'deopping' themselves (using "-o").

  Numeric Replies:
  ERR_USERSDONTMATCH
  RPL_UMODEIS
  ERR_UMODEUNKNOWNFLAG

  Examples:
  :MODE WiZ -w ; turns reception of WALLOPS messages off for WiZ.
  :Angel MODE Angel +i ; Message from Angel to make themselves invisible.
  MODE WiZ -o ; WiZ 'deopping' (removing operator status). The plain reverse of this command ("MODE WiZ +o") must not be allowed from users since would bypass the OPER command.

- This document is currently a draft, more commands will be added and existing commands will be expanded

Deliverable Schedule

- Register an account on Gitlab.com and send me screenshot of your user details. Specifically, I need to see your @username to add you to the class folder
- Create client and server applications that allow multiple clients to connect to a server with each connection on its own thread
- Build support for several data structures in memory including for tracking channels and connected clients
- Extend application to support message passing

  - Client should be able to send message to server
  - Server should be able to broadcast message to all clients
  - Client should be able to receive message and print it even if client is asking user for input

- Build support for threaded file I/O including for logging and reading in and updating configuration files
- Build support for command line arguments that override configuration file settings, including the display of a help message
- Build support for server commands as expressed in this document *(note: document is not complete yet)*
- Maybe... Extend Client/Server model to allow multiple servers with their own connected clients and messages propagating through the network.

Final Notes:

- Ensure the source code is well written. Take pride in this project.
- Separate source files based on function and readability
- Use header files to organize function/struct declarations and constants
- Use a Makefile to simplify the compilation and testing process
- Use proper variable names
- Everything that is a changeable decision needs to have a hard coded default, overridable from the config file, overridable from command line arguments.
- Document your code, your project, your decisions. How to use your program, how to use your config file, etc.