Features

Server

- The core of the server uses epoll API and nonblocking IO.
- The server has a map of each open socket to some user data.
- The server also has a map of username to nicks which is loaded and saved to a file.
- The server uses buffers in the user data struct to receive/send messages.
- The server currently supports the following commands from the client: MOTD, NICK, USER, PING, QUIT
- The user data uses a message queue that allows the server to prepare multiple messages to send to one client.
- The message queue is also useful for message chat as messages for a target user can be pushed to their respective queues.

Logic

The main logic is that the epoll listens for Read/Write events on all the available client connections as well as the server listening socket. If the event is on the listening socket, that implies the server can connect to new clients and initialise their user data. This is done through the `Server_process_request()` function. In the case the event is on a client socket, I handle three cases: read, write and error. On error, I disconnect the client. On write, we send any pending messages from that user's message queue. Lastly, on read, we receive any data and parse the message if it is complete. Otherwise, we store the bytes in the user's buffer for later.

There are various functions to handle the commands in the form of `Server_reply_to_XXXX()`. These functions use the predefined message reply strings in `reply.h` and substitute the reply parameters such as user's nick. The message parser in `message.h` is used to parse message details safely.

QUIT

The QUIT command is used by a client to indicate their wish to leave the server. All data associated with this user is freed and socket is closed. An ERROR reply is sent before closing the socket to allow the reader thread at the client to quit gracefully. This is done through a 'quit' flag that is set to true when the QUIT request comes. When the final message to the client is sent and if a quit flag is seen, the server knows to close that connection at that point.

MOTD

This command sends a Message of the Day to the requesting user. It looks up the current day's message from a file (motd.txt) that contains a list of quotes. This file was downloaded using the `zenquotes.io` API using the script

in `download_quotes.py`. This list can be updated by repeating this script. The server simply gets the line that is at position `current_day % total_lines`. The filename can be changed at any time as the server has a string to store the filename.

(NICK/USER) User Registration

- User can register with NICK and USER commands
- NICK can be used to update nick at any time
- Server keeps track of all the nicks that have been used by users
- User may skip NICK to reuse previous NICK if there is one
- USER can only be used once at the start to set username and realname.
- username is private to the user, it is the main identification source
- Users can use NICKs to send messages to another user
- User can use any NICK that is owned by a user

Client

The client implements a thread-based model with four threads:

Client Outbox thread

- The outbox thread blocks if the outbox queue is empty.
- This thread pulls messages from the queue and sends them to the server over existing tcp connection.
- If a 'QUIT' message is discovered this thread will quit immediately after sending it to the server.

Client Inbox thread

- The inbox thread will block if the inbox queue is empty.
- The inbox thread pulls messages from the inbox queue and handles them.
- Most commonly the messages are printed to stdout for the user to see.
- On an ERROR message the inbox thread will quit immediately. The ERROR message is also a reply to a QUIT message.

Client Reader thread

- The reader thread will asynchronously read() from the server using the epoll API.
- This thread will use a buffer to store incomplete messages.
- It will handle the case if there are multiple messages sent at once.
- It will add each message to the inbox queue for the inbox thread to handle.
- It will never block on an enqueue as the queue has no maximum size.

Client Main thread

- It is the duty of the main thread to read user input from stdin.
- This thread blocks on the getline() instruction.
- If the input is a valid IRC command, the string will be terminated by CRLF appropriately.
- If the input is a special command starting with a /, the client will generate the corresponding command.
- The irc command is added to the outbox queue to send to the server.