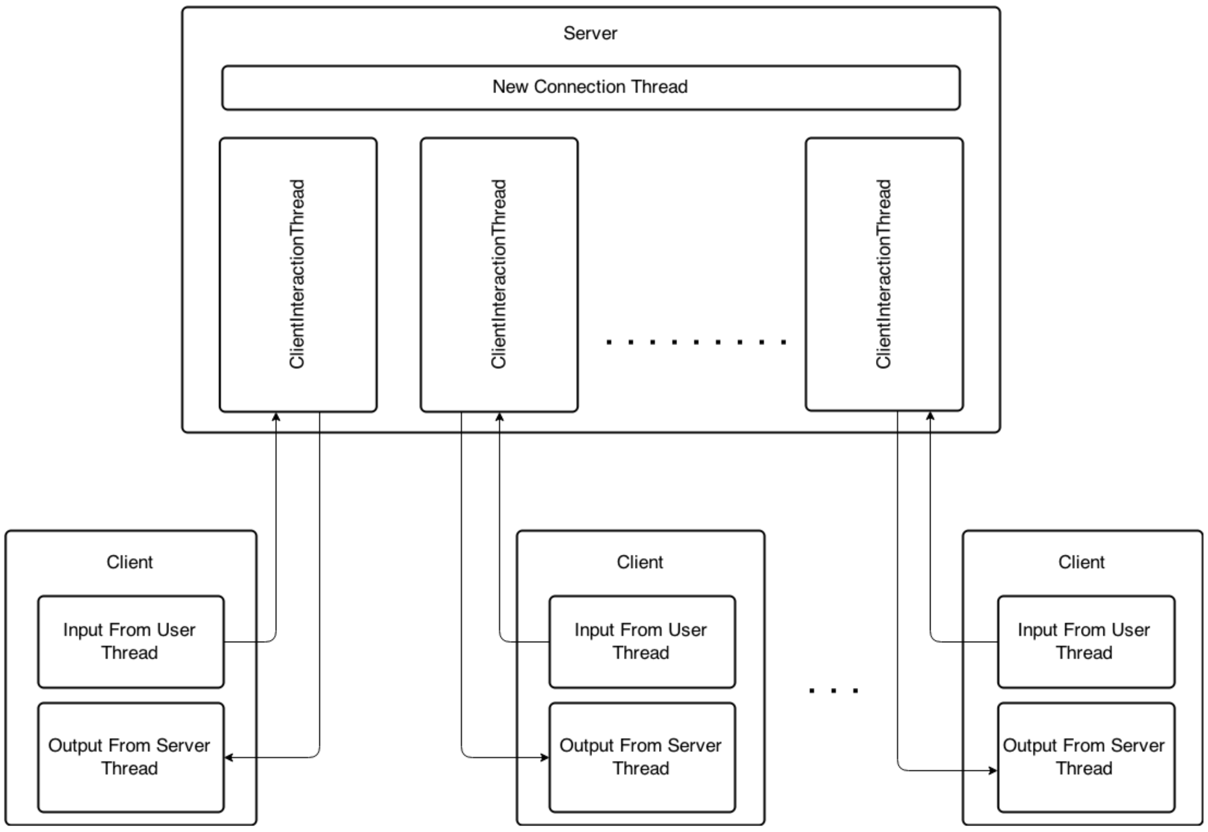
Matt Gautreau
Zach Montoya
CS 422
Final Project Report

1 Overview

MYRC is a Java-based distributed chat application which seeks to implement a subset of the IRC protocol as faithfully as possible. The implementation includes both server and client programs, and facilitates client-server, and client-client(s) communication through the server. This communication is done using sockets.

2 System Architecture

The general structure of the system is as follows:



2.1 Server Architecture

The IRC protocol specifies that servers manage a set of chatrooms, called channels, which have the following fundamental properties:

1. Name - usually indicative of the topic of the channel
2. Topic - a brief explanation of the channel's purpose
3. Users - a set of users currently in the channel

Users must be able to send messages both to other users and to an entire channel. For this purpose, the server maintains a collection of all channels it is currently managing, as well as the collection of all users that are connected to the server.

The main server thread, labeled “New Connection Thread” in the architecture diagram, simply exists to constantly wait for a connection to the `ServerSocket`. When a new client connects, we create a socket assigned to that particular client, a `UserInfo` object associated with the connecting user, and then start a `ClientInteractionThread` to handle server-client communication with the new client.

The `ClientInteractionThread` will constantly wait to receive a message from the client, parse that message, and perform the action associated with the message. Any responses that are generated while executing the message will be sent back to the client.

2.2 Client Architecture

On startup, the client will connect to the server via the server’s new connection thread. Once a connection is established, the client will create two threads. One thread will be waiting for user input (in the form of an IRC message) and sending it over its socket to its corresponding `ClientInteractionThread` on the server. The other thread will be responsible for receiving chats or other responses from the server, and printing them so the user may see them on `stdout`.

2.3 Message Handling

MYRC currently supports the following IRC commands. Since our implementation is limited, each command will contain a description of the precise way in which it is handled in our system.

- PASS
  Description: The first of two commands to register a user.
  Syntax: PASS <password>
- NICK
  Description: Sets the unique nickname for the client.
  Syntax: NICK <nickname>
- USER
  Description: The second of two commands to register a user. Sets the realname for the client and the unique nickname for the client. The hostname and servername arguments are ignored for now.
  Syntax: USER <nickname> <hostname> <servername> ”:”<realname>
- JOIN
  Description: If the target channel exists, the user joins the specified channel and receives all updates from the channel. If JOIN is followed by ”0”, the client is removed from all of the channel they have joined.
  Syntax: JOIN (<channel>*(”,”<channel>)) / ”0”
- PRIVMSG
  Description: Allows the user to send a message to another user or channel, if the msgtarget exists.
  Syntax: PRIVMSG <msgtarget> ”:”<text to be sent>
- TOPIC
  Description: Allows the user to get the topic of a channel if topic is not set. Allows the user to set the topic of a channel if the topic is set.
  Syntax: TOPIC <channel> [<topic>]
- PART
  Description: Allows the user to leave the channel(s) that they had already joined. The user may display a custom message upon leaving.
  Syntax: PART <channel>*(”,”<channel>) [”:”<Part Message>]

The message handling system is as follows. The raw string of a message has an associated grammar. We use this grammar to parse the message, which then contains a `command` field, in addition to components called `parameters`, `prefix`, and `trailing`. The `command` field is used to determine which of the above message types it is. We have an abstract class called `Message`, which contains all the fields of the message, and includes an abstract method called `executeCommand`. We then create extensions of the abstract class for each of the supported commands, implementing the `executeCommand` in accordance with the behavior of the message as defined in the IRC protocol.

3 Compiling and Running

3.1 Creating the Executables

Currently the server and client are both packaged together. The following targets are recognized by the Makefile

`make all` will compile both the server and client.

`make runServer` will start the server.

`make runClient` will start a client and connect to the server (the server must be running first).

3.2 Running the Client

When running the client program, the client must first register their connection. The steps to register a connection is as follows:

1. PASS command
2. NICK command (optional)
3. USER command

Once the client's connection is registered, they are allowed to submit any of the other implemented commands, with the exception of PASS and USER because they may be only used during connection setup. Additionally, it should be noted that the default channels that our IRC server establishes are `#Sports`, `#Politics`, and `#HackerNews`

3.3 Example Message Sequence

1. `PASS secretpassword`
2. `USER zacharymontoya examplehost exampleserver :Zachary Montoya`
3. `JOIN #HackerNews`
4. `TOPIC #HackerNews :A place to stay up to date on current technologies`
5. `PRIVMSG #HackerNews :Is anyone there?`