CS 594 Thomas Lancaster and Patrick Rademacher
Internet Draft Portland State University

Intended status: IRC Class Project Specification December 6, 2019
Expires: June 2020

Internet Relay Chat Class Project
pat_chat_rfc.txt

Status of this Memo

This Internet-Draft will expire on June 6, 2020.

Abstract

This memo describes the communication protocol for an IRC-style client/server system for the Internetworking Protocols class (CS494/594) at Portland State University.

Table of Contents

1. Introduction
2. Conventions used in this document
3. Basic Information
4. The Server
5. The Client
6. Chatrooms
7. Security Considerations
8. Conclusion & Future Work
9. IANA Considerations
10. Normative Considerations
Author’s Emails

1. Introduction

This specification describes a simple IRC (Internet Relay Chat) protocol called Pat Chat. Clients can connect to a central server and communicate with each other via text messages. The relay in the IRC name refers to the central server that ‘relays’ messages between connected clients.

Users can join rooms and any client that messages in the room will have their message shared with all other connected clients in that room on the regular default settings. However, a client can also

broadcast messages across multiple rooms and choose to not broadcast it to whatever room they are currently part of.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. Basic Information

The Pat Chat server runs on port 8083 by default. This can be changed as necessary for different configuration and network needs. All communication described in this protocol takes place over TCP/IP networking protocols. A persistent TCP connection is maintained between the client and the server. The client can send messages through this connection and the server can also push messages to the client.

The protocol is plain-text-based which is done for ease of programming between client and server, and also for human readability and understanding. It is also asynchronous meaning the client and server communicate with each other whenever desired. They MAY choose to send an error message to the other party informing them of the reason for connection termination. Other error messages are provided to indicate when either the server or client’s program has crashed.

The server as currently constituted can serve up to 100 users and, in theory, an infinite amount of rooms but can be configured to meet various network demands.

4. The Server

4.1. To make the overall program work, the server file must be started before any client file. This is because the socket must be created from the server end for a client to connect. This is executed by doing the following in the command line:

python3 server.py

4.2. The server then boots up, where it obtains its IP address using the socket library for Python. From there, it exports the IP address to a .txt file that the client will then import to establish a connection.

4.3. The server then listens asynchronously for clients to connect. When a client connects, the server will accept it, and from there, stores the client's connection information and IP address in two respective lists: list_of_clients and list_of_IP. These lists are used to keep track of clients' information, including their username (further notation forthcoming). When a client disconnects, these lists are used to symbolize and demonstrate that a connection has been lost.

4.4. Once a client has established a connection, the server takes the client through a login process. There is a .txt file that stores all previous user information. If the user chooses to use an existing login, this information is pulled and ensures a match for a username as well as that username's password. Otherwise, the option for creating a new user is available, where this information is then added to the .txt file for future use.

4.5. If the user login is successful, the server then creates a new thread for that specific client. This creates an asynchronous loop for each client in which the server needs to communicate with. All of the remaining functions of the program (in other words, everything aside from login functions and exit functions) take place within this thread.

4.6. The following functions all occur within the thread (details of them are discussed later):

1. clientthread(connection, address, username) -- this is the main function that is called in the thread.
2. main_menu (connection, address, username, currently, boc (broadcast only current), bts (broadcast to selected)

3. broadcast(message, connection, address, username, currently, boc, bts)
4. remove(connection, address, username)
5. create_a_chatroom(connection, address, username)
6. join_a_chatroom(connection, address, username)
7. client_crash_test(response, username)
8. client_crash_test_login(response)
9. list_people_in_chatroom(connection, address, username)
10. list_all_chatrooms(connection, address, username)
11. remove_user_from_chatroom(connection, address, username)

5. The Client

5.1 After starting the server, multiple clients can initiate sessions and run simultaneously. New clients execute client.py script to begin.

client.py:

When a new client starts the script and runs, the following happens:

5.2 Each client possesses a unique IP address that is bound to the socket of the server (8083). When the client connects to the server, the server streams the client's data to the event loop handler. The client is then added to a list of active clients, a list of active IP addresses and a list of client information.

5.3 Within *server.py*, there is a user class that contains the socket connection and address of the client. This class has a login method that a user uses to sign in with their username and password. The client listens to and sends messages to the server using an asynchronous gather handler which moves the client to the main menu.

5.4 The client waits for messages to arrive from the server in an infinite loop. These can be messages from the server who is responding to a client command or individual or broadcast messages from other clients attached to the server.

5.5 Within this infinite asynchronous loop, the client can send simple messages to the server, or it can issue one or many commands that are handled by a response list.

5.6 In the event of a server crash, the loop recognizes the stream of messages is no longer present, notifies the client that the server is disconnected, and closes the client connection and program.

6. Chatrooms

6.1 Available Functions: The main functions of Pat Chat are to create a room, join a room, switch to a room, and leave a chat room. When the user initially logs in, they are greeted by a message that instructs them to the “Main Room.” Each client is then given a toolbox of commands to navigate throughout Pat Chat.

6.2 Active Users in a Room: Any client can see a list of active users in a room by typing “l”. This function does not display all active users in Pat Chat, but this could be added in later versions.

6.3 Create and Join a Chatroom: Any client can create a chat room by typing “c”. If the room name is accepted, then the room is created successfully and they are automatically a member of the room.

Any active member within Pat Chat can enter a newly created room by typing “j” and the room name. They can join multiple rooms. They cannot, unfortunately, see the chat history of the room up to the moment of joining. Future functionality could include printing all rooms and current users.

6.4 Leave a Chatroom: A user can leave a chatroom by sending the remove command “r” and the room name. This will remove them from the list of clients that have joined the room. They also cannot enter the room unless they join again.

6.5 Exit Pat Chat: The user may exit the program by entering “exit” at any time. This action closes the socket stream and an exit message is printed to the screen. The program will proceed to exit.

7. Security Considerations

Any messages sent through Pat Chat are in plain-text and unencrypted. User account info is also stored in a plain-text file. If secure communications required, the end-user must encrypt data before transport. User account info could be stored on a third-party database for enhanced security purposes.

8. Conclusion & Future Work

Additional features and commands of Pat Chat could be added at a later date as demand requires. The goal of the project is to have a functioning IRC style client and server model. More dynamic features could also be added such as a GUI.

9. IANA Considerations

None

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Authors' Emails:

Thomas Lancaster - tlan2@pdx.edu
Patrick Rademacher - prad2@pdx.edu