CS494P Internet Protocols
Internet Draft
Intended Status: IRC Class Project Specification
Expires: December 1st 2020

Orion Crocker
Portland State University
June 1st, 2020

Internet Relay Chat Project

Status of this Memo

This document specifies the requirements, functionality, and usage of SimpleIRC, an internet relay chat program created as a project for CS494P Internet Protocols. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Copyright Notice

Abstract

SimpleIRC is an internet chat relay program written in Python as a project for CS494P Internet Protocols at Portland State University, Spring 2020.

Table of Contents

1. Introduction

SimpleIRC is a simple internet relay chat protocol by which clients can communicate with one another via text. This system employs a central server which ‘relays’ messages that are sent to it to other connected users.

Users can join individual rooms, in which users within that room are subject to the same data stream. Any message sent to that room is forwarded to all other users currently joined to that room. All conversations inside of rooms are logged by the server.

Additionally, users can send private messages to one another. Direct messages (DM) messages are never logged by the server.

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].

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

3. Basic Information

All communications described in this protocol take place over TCP/IP, with the server listening for connections on port 2020. Clients connect to this port and maintain this persistent connection to the server. The client can send messages and requests to the server over this open channel, and the server can reply via the same. This messaging protocol is inherently asynchronous - the client is free to send messages to the server at any time, and the server may asynchronously send messages back to the client.

As described in [4.2], both server and client can terminate the connection at any time for any reason. They MAY choose to send an error message to the other party informing them of the reason for connection termination.

The server MAY choose to allow only a finite number of users and rooms, depending on the implementation and resources of the hostsystem. Error codes are available to notify connecting clients that there is currently a high volume of users or groups accessing the server.

4. Client Messages

4.1. First message sent to server

```
sock.send(name.encode())
```

4.1.1. Usage

Before messages can be sent, a connecting client MUST provide a chat name and identify which version of the protocol they are using.

The server MUST associate the client’s chat name with the socket connection of the server. This message SHOULD be sent only once; if the server receives the message more than once, the server MAY either ignore the message or terminate the client’s connection.

4.1.2. Field Definitions

- sock: TCP connection between client and server using Python's socket library.
- name: Specifies the name the connecting client wishes to publicly identify as. If the name already exists, the server defaults to the connecting IP address as a temporary name. A new name can be selected after joining.

4.2. Listing Rooms

```
\ls rooms
```

4.2.1. Usage

Command is sent by the client to request a list of all the rooms currently available to join, as well as all rooms the user is currently in.

4.2.2. Response

Server MUST return a list of all rooms and currently joined rooms as a byte string message to the requesting client.

4.3. Joining and Creating Rooms

```
\join room name
\create room name, room description
```

4.3.1. Usage

Sent by the client to join a chat room. If no room by that name exists, the client has the option to create such a room.

Upon joining a room, the server MUST send a message to all users currently in that room to alert current room occupants that the user list has changed. The identifier MUST be set to the name of the room. Upon leaving a room, the server MUST send a message alerting current room occupants that the user list has changed.

4.3.2. Field Definitions

- \join: Client command to join specified room.
- \create: Client command to create a new room.
- room name: Name of the room to join or create.
- room description: Message sent to the user when joining a room. This can be a greeting or a short description of the intended topic of conversation.

4.4. Leaving a Room

\leave room name

4.4.1. Usage

Sent by the client to leave the chat room.

Upon receiving this message, the server MUST remove the client from the specified room and MUST inform the other members of the room that the user list has changed. The server SHOULD ignore leave requests when the client is not currently a member of the specified room.

4.4.2. Field Definitions

- \leave: Client command to leave specified room.
- room name: Name of the room to leave.

4.5. Sending Messages

sock.send(data)

4.5.1. Usage

Sent by a client to send a text message to either a room or another user. If the message's target recipient is valid, the server MUST relay the message to the target recipient. If the message's target recipient is invalid or does not exist, the server will ignore the request.

4.5.2. Field Definitions

- sock: TCP connection between client and server using Python's socket library.
- data: Client textual message as a byte-encoded string. If a message is longer than 1024 bytes, the remainder is ignored.

5. Server Messages

5.1. Forwarding Messages to Clients

connection.send(message.encode())

5.1.1. Usage

Server sends each message directly to the socket connection associated with the client. Server MUST NOT send messages to clients which are not associated with another client's connection.

5.1.2. Field Definitions

- connection: Established TCP/IP connection between server and client using Python's socket library.
- message: A message sent from the server to the client machine.
- encode: All messages sent between server and client are byte-encoded strings.

6. Error Handling

Both server and client MUST detect when the socket connection linking them is terminated, either when actively sending traffic or by keeping track of the heartbeat messages. If the server detects that the client connection has been lost, the server MUST remove the client from all rooms to which they are joined. If the client detects that the connection to the server has been lost, it MUST consider itself disconnected.

As previously stated, it is optional for one party to notify the other in the event of an error.

7. Conclusion and Future Work

This specification provides a generic message passing framework for multiple clients to communicate with each other via a central forwarding server.

Without any modifications to this specification, it is possible for clients to devise their own protocols that rely on the system of textual relay described in this document. For example, clients may choose to encrypt their traffic using PGP. Additionally, the current infrastructure could be used to distribute files, or even live video information between clients.

8. Security Considerations

Messages sent using this system have NO protection against inspection, tampering, or outright forgery. The server sees all messages that are sent through the use of this service. 'Private' messages between clients may easily be intercepted by a 3rd party that is able to capture network traffic. Users wishing to use this system for secure communication should use/implement their own user-to-user encryption protocol. Additionally, all messages sent bound for public chat rooms are logged by the server.

9. IANA Considerations

None

9.1. Normative References

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