Dakota Sanchez
Portland State University
June 2017

IRC project specification
Expires: Never

Internet Relay Chat: Client and Server Protocols

Status of this memo

This memo provides information for the Internet community. It is not an Internet standard. This memo may be distributed freely.

This memo may be amended in the future.

Copyright Notice

Abstract

This IRC (Internet Relay Chat) protocol specification can be used to implement a text-based chat-room style application. The client section specifies how to communicate with the server, and vice versa.

Table of Contents

1. Introduction

This version of the IRC protocol is a simplified version of the original IRC protocol. Only basic chat-room functionality is specified.

This protocol is intended to work by having a central server for clients to connect to. Messages that are sent to the server are relayed to other clients in the same virtual room.

1.1 Clients

In the context of this protocol, clients refer to programs making a UNIX-style socket-based TCP connection to a persistent server program on the same or possibly different computer on the same local area network. This clients can send and receive messages as well as manage their own state if they wish.

1.2 Server

In the context of this protocol, a server is a persistent program running on an agreed-upon port waiting for TCP socket connections. When a connection is made, the best practice is to spawn another socket to handle the session with the new client, and continue to listen on the main socket for new connections. The server manages all of the connections as well as incoming/outgoing messages.

2. Overview

The connections between client and server should use the TCP protocol and use port 10000 on the server-side. Individual connections to clients on the server-side can be on any open port. The messaging between clients and servers is meant to be asynchronous. Both sides can send messages whenever they want, and should be listening for incoming messages on a separate thread.

Both ends of the connection can be terminated at any time. The server and client should be prepared to clean up network resources and gracefully exit whenever this happens.

3. IRC concepts

In IRC applications, you join/leave virtual rooms and can send messages to other participants in the same room. You can also join multiple rooms and toggle between them. You can also create rooms. Other common features include listing available rooms as well as their occupants.

4. Message Overview

Most of the message logic happens server-side, so the client message formats are pretty simplistic.

4.1 Basic Format

Client messages:

```
-----------------------------
|room or command ‘ ‘ message|
-----------------------------
```

The client sends a space-delimited message to the server. The first token can either be a command or a room name. The server decides that it is a room name if it’s not in the list of commands. The rest of the message contains additional information for the server to execute the command, or a chat message for other clients.

Server messages:

```
----------------------
|command|room|message|
----------------------
```

The server sends a pipe-delimited message in its TCP packets. The room and message portions are optional and only used depending on the command. The command specifies what was executed on the server (create, join, etc.), the room specifies what room was operated on, and the message has additional info or a chat message from another client.

```
Server message codes:
     fff0: room membership
     fff1: room does not exist
     fff2: room list
     fff3: room created
     fff4: room joined
     fff5: room left
     fff6: chat message
     ffff: error in last command
```

4.2 Error messages

The only error messages come from the server when it fails to execute a command given to it by a client. It will send the command ‘0xFFFF’ to signify an error has been encountered in the last command.

5. Messages

5.1 Connection

There is no explicit message for connection establishment. That is left up to the TCP sockets. When a connection is made on the server side, the listening socket will spawn a new session and this can be printed to the terminal.

5.2 Listing rooms

```
Client:
     ------
     |list|
     ------

Server:
     ------------
     |fff2|rooms|
      -----------
```

5.3 Listing room membership

```
     Client:
          ---------------
          |list ‘ ‘ room|
          ---------------

     Server:
          -----------             -------------------
          |fff1|room|     or      |fff0|room|members|
          -----------             -------------------
```

5.4 Creating rooms

```
     Client:
          -----------------
          |create ‘ ‘ room|
          -----------------

     Server:
          -----------
          |fff3|room|
          -----------
```

5.5 Joining rooms

```
     Client:
          ---------------
          |join ‘ ‘ room|
          ---------------

     Server:
          -----------
          |fff4|room|
          -----------
```

5.6 Leaving rooms

```
     Client:
          ---------------
          |leave ‘ ‘ room|
          ---------------
     Server:
          -----------
          |fff5|room|
          -----------
```

5.7 Sending messages to rooms

```
     Client:
          ------------------
          |room ‘ ‘ message|
          ------------------

     Server:
          --------------
          |fff6|message|
          --------------
          (message contains id of sender so receivers know)
```

5.8 Error

```
     Server:
          ------
          |ffff|
          ------
          (generic error-in-last-command message)
```

6. Handling lost connections

The server should be able to handle lost clients due to its mutli-threaded nature. Any lost connections can be forgotten as long as the room membership is updated.

The client should be able to handle a lost connection as well, easily identified by an empty response from the TCP socket. A message such as ‘Connection to server lost.’ can be displayed and the connection can be cleaned up.

7. Security considerations

This version of the protocol offers no security guarantees, but a future version easily could if the server and client used some sort of token-based authentication and message encryption. This could prevent false logins as well as man-in-the-middle attacks.

8. Contact

dakota@dakotasanchez.com