Internet Relay Chat Class Project
Draft-irc-pdx-cs594-spring.txt

Status of This Memo

This Internet-Draft is used for the author’s personal use. It is not ready for publishment.

Abstract

The Document helps users understand the communication protocol for an IRC-style client/server application for the Internetworking Protocols class at Portland State University.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

This specification describes a simple IRC protocol by which clients can communicate with each other. The server receives messages from clients and distributes them over the users that are in the same rooms.

Users can create rooms, which are groups of users that are subscribed to the same message stream. Besides, users can join and leave rooms. Joining rooms can be done simultaneously.

The utility functions are to list all user in a specific room, to list all rooms in the server, and to list all rooms that a user are being in.

2. Conventions used in this document

The keywords "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.

3. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 55000. 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.

The server and client may terminate the connection at any time for any reason. They MAY choose to send an error message to other part informing them of the reason for connection termination.

The server MAY choose to allow only a finite number of users depending on the implementation and resources of the host system.

4. Communicating Entities

4.1 Server

The Server MUST start first. Server keeps a list of rooms. It is a message hub. Every message must be received and distributed to room destination by Server. The data structure of the list of rooms is the dictionary. Dictionary makes the program avoid looking up the room id whenever we need to access to a room entity.

4.2 Room

Rooms are created by Server. It has a list of users who participate in the same room. Room entities are created by the server application and managed by Server entity. The rooms are responsible for sending messages to users. This helps server distinct messages from different rooms. The data structure of rooms is list/array. The room entity does not access to user object often, so using list/array helps the program save memory.

4.3 User

User are created in the server application. They are managed by Room entity. The users' identification is usernames, every user is REQUIRED to provide a username. Users entities also have information about their sockets (addresses and socket objects). The Users entity manages a list of the id of rooms that this user is in. The list will be used to check if users have the ability to send messages in a room. The Clients will communicate with Server using IP and Port numbers through User entities. The data structure of the list of room id is dictionary because we need to access to find the room id very often.

5. Server Messages And Events Handling

“Welcome new user.”: This message welcomes new users to the server. The user is REQUIRED to provide a new username. After providing a username, the user is ready to use the chat application.

“#exit”: The server sends this message to inform that the user is removed from chat rooms and the communication socket is closed. After receiving this message, the client application is ready to close.

“invalid room name”: the clients provide an empty room name to the server.

“room does not exist”: the room operation can be done because the server cannot find the room id.

“you are not in any room”: the server cannot find the user in any room in the server.

“you are not in the room”: the room operation is declined because it is only allowed when the user is in this room.

6. Client Messages And Events

“#username”: This message MUST have two parts. It tells the server that the user is new and the value comes after this keyword is the name of the user. This is REQUIRED for every user when the clients first connected to the server.

“#ls”: This message does not require extra parts. It requests a list of rooms from the server. The server will send back a list of room names (room id) with the number of users being in the rooms.

"#mk": This message MUST have two parts. It tells the server to create a new room with the room id provided which is the second part of this message. The room id MUST NOT be replicated. After creating the new room, the server automatically puts this user into the room just created. If the room id already exists, the server sends back "room name is taken". If the room id is empty, the server sends back "invalid room name".

"#join": This message MUST have two parts. The second part MUST be the valid room id. The client tells the server to put it in the room. If the room id does not exist, the server sends an error message back to the client.

"#leave": This message MUST have two parts. The second part MUST be the valid room id. The client tells the server to remove it in the room. If the room id does not exist, the server sends an error message "room does not exist" back to the client. If the room id is empty, the server sends "invalid room name" back to the client.

"#to": This message MUST have three parts. The second part MUST be the valid room id. The third is the message. The client tells the server to remove it in the room. If the room id does not exist, the server sends an error message "Room does not exist" back to the client. If the room id is not found in the list of room id in the User entity, the server sends an error message "You are not in the room" back to the client. If the room id is empty, the sends an error message "invalid room name".

"#myrooms": This message does not require extra parts. It requests the server to send a list of rooms that this user is in. If this user is not in any rooms, the server sends back a message "You are not in any rooms.". Otherwise, the server will the list.

"#users": This message MUST have two parts. The second part MUST be the valid room id. The client tells the server to list all users in the room. If the room id does not exist, the server sends an error message "room does not exist" back to the client. If the room id is empty, the server sends "invalid room name" back to the client.

"#help": This message does not require extra parts. It requests the server to send the list of the user interface. There is not any extra content besides the list.

"#exit": This message does not require extra parts. It requests the server to remove the user from all the rooms. After removing, the server sends back "#exit" back to clients.

"#crash": This message does not require extra parts. This message is automatically generated by clients when the KeyboardInterupt is executed. It tells the server that the client is crashed and the server MUST remove the user from the rooms. There is no message sent back to the clients.

7. Event Handling

Sending messages: The server receives the request message along with the room id and the derivable message. The server looks up for the room id and calls the appropriate room entity to broadcast the message. The message is delivered by room entity, so the message cannot be sent to the wrong room. Because the room has a list of users, the mistake of sending a message to the wrong users is prevented. In the reply message, the server specifies who sent the message and from which rooms. This helps users distinguish different messages.

Receiving messages: Because the correctness is handled at the server, the clients only just receive the messages through the communication sockets repeatedly.

8. Error Handling

Server crash: the crash will be detected if the message from the server is NULL. In this scenario, the client application will be terminated.

Client crash: the crash will be detected if the message from the client is NULL. The client will send the message “#crash” to the server. The server will remove the client from its rooms, close the sockets, and keep running.

ERROR_SOCKETS: the error information is created by the select.select() function. Both client and server applications have this error. When they receive this error, all the communicating socket will be closed.

9. Conclusion & Future Work

The application fulfills all the basic functionality of IRC. However, the server censors all the messages from users. This is a security concern. To implement a secure chat, the developers must implement it.