CS594
Internet Draft   Portland State University
Intended status: IRC Class Project Specification November 29, 2022
Expires: June 2023

Internet Relay Chat Class Project

Draft-irc-pdx-cs594-00.txt

Status of this Memo

Copyright Notice

Abstract

The project depends on the concept of Internetworking Relay Chat. Through this project we make an attempt to achieve communication between server and client. It involves a user using the functionalities listed below:

1. Creating a room
2. Joining a room
3. Exit a room
4. List of members in a room
5. List of rooms created
6. Chat in a room
7. Get Help
8. Error Display

It enables multiple clients to communicate with each other over a single server. For instance, let us consider the online game Game ‘Ludo’. The players first enter a room with a code or create a room which generates a code in order to start a game. It displays all the members joined into that room. During the game, it enables the users to chat via a chat box which can be read by all the users in that particular room. If a user’s internet is down, it throws a message ‘Unable to connect to the server’. Similarly, we try to achieve the same concept in this project.

Table of Contents

1. Introduction

The primary aim of Internet Relay Chat project is to build a game which involves chat functionality protocol which enables the users to communicate with each other. It comprises of a central server which relays messages that are sent to it to the other users connected. Users would be able to create rooms, join rooms, chat with one another, exit the room and Get Help.

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 server is open for connections on port 31425, and so this protocol only enables TCP/IP for all communication. This persistent connection is maintained by clients connecting to this port. relationship with the server. Over this, the client can interact with the server by sending messages and requests. open channel, and the server can respond in a same manner. It's inherent in this transmission protocol Asynchronous means that both the client and the server are free to receive and transmit messages at any time. back to the client asynchronously with messages. The client as well as the server may both end Any time, for any cause, a connection can be made.

The two could decide to communicate an error to one another. party informing them of the connection's termination's cause. Depending on the configuration and resources of the host system, the server may decide to limit the number of users and rooms that can be created. The number of client connections is set to six for the present project.

4. Message Infrastructure

The client delivers the orders to be executed in the format command argument1 argument2, where command is a four-letter word that can be written in any case (uppercase, lowercase, or a combination of both). When a user's addition is inaccurate, errors are taken into consideration in the front of the response message. But no errorcode is needed for this straightforward system. The server only sends the client a string representation of the error message or response.

4.1. Generic Message Format

```
struct irc_pkt_header  {
       uint32_t opcode;
       uint32_t length;
   };

   struct irc_pkt_generic {
       struct irc_pkt_header header;
       uint8_t payload[header.length];
   };
```

4.1.1. Field definitions:

- o header.opcode - specifies what kind of message is contained within the payload.
- o header.length - specifies how many bytes of payload follow the header in this message (excludes header size).

- o The length must be verified by the client and the server to be appropriate for the given opcode. Otherwise, the entity that notices the error MUST break the connection and MAY inform the other party of the error by sending them the message IRC ERR ILLEGAL LENGTH (see error messages section).
- o payload - variable length payload. Not used by some messages.

4.1.2. Operation Codes (opcodes)

```
       IRC_OPCODE_ERR             = 0x10000001
       IRC_OPCODE_KEEPALIVE       = 0x10000002
       IRC_OPCODE_HELLO           = 0x10000003
       IRC_OPCODE_LIST_ROOMS      = 0x10000004
       IRC_OPCODE_LIST_ROOMS_RESP = 0x10000005
       IRC_OPCODE_LIST_USERS_RESP = 0x10000006
       IRC_OPCODE_JOIN_ROOM       = 0x10000007
       IRC_OPCODE_LEAVE_ROOM      = 0x10000008
       IRC_OPCODE_SEND_MSG        = 0x10000009
       IRC_OPCODE_TELL_MSG        = 0x10000010
       IRC_OPCODE_SEND_PRIV_MSG   = 0x10000011

       IRC_OPCODE_TELL_PRIV_MSG   = 0x10000012
```

4.2. Error Messages

```
   struct irc_pkt_error {
       struct irc_pkt_header header =
           { .opcode = IRC_OPCODE_ERR, length = 4 };
       uint32_t error_code;
   }
```

4.2.1. Usage

Prior to terminating a TCP connection, MAY be sent by either the client or the server to let the other party know why the connection is being terminated. Any client or server that receives this message SHOULD assume that the connection.

4.2.2. Field definitions:

o error_code - specifies the type of error that

occurred 4.2.3. Error Codes

```
       IRC_ERR_UNKNOWN         = 0x20000001
       IRC_ERR_ILLEGAL_OPCODE  = 0x20000002
       IRC_ERR_ILLEGAL_LENGTH  = 0x20000003
       IRC_ERR_WRONG_VERSION   = 0x20000004
       IRC_ERR_NAME_EXISTS     = 0x20000005
       IRC_ERR_ILLEGAL_NAME    = 0x20000006
       IRC_ERR_ILLEGAL_MESSAGE = 0x20000007
       IRC_ERR_TOO_MANY_USERS  = 0x20000008

       IRC_ERR_TOO_MANY_ROOMS  = 0x20000009
```

4.3. Keepalive Messages

```
   struct irc_pkt_keepalive {
       struct irc_pkt_header header =
           { .opcode = IRC_OPCODE_ERR, .length = 0 };
   }
```

4.3.1. Usage

provides an application-layer notification of a disconnected peer.

MUST be sent by the client and server at least once every five seconds to let the other side know that they are still connected.

If more than a predetermined amount of time has passed after receiving one of these keepalive signals, the client and server SHOULD consider the other party offline. If such a timeout is employed, the duration MUST be longer than 15 seconds.

5. Label Semantics

Sending and receiving labels is necessary for identifying users and rooms. All label rules must be validated in conformity with the following criteria:
• Must only be composed of readable UTF character values.
• The input size must be at least 1 character and no greater than 100k.

Sending and receiving labels is required for identifying users and rooms. The following validation criteria MUST be applied to all label rules:

o The field size for transmission is always exactly 20 bytes.

o Must only include readable ASCII character values between 0x20 and 0x7E.

o The length must be at least 1 and a maximum of 20 characters. o Spaces are not allowed at the beginning or conclusion of a sentence.
o If the message is less than 20 characters, the next character MUST BE A NULL CHARACTER (0x00). The remaining characters COULD potentially be null characters.

o The receiver MUST cut off the connection if any of these rules are breached, and it MAY issue the IRC ERR ILLEGAL NAME error.

To lessen the possibility of a buffer overflow attack, receiver SHOULD append a NULL terminator to this array before utilizing this field.

6. Client Messages

On the client side, the user will send particular commands with the arguments to the server.

a. Welcome client
   Usage: person will enter his username
   Response: welcome user
b. List all rooms
   Usage: person will enter command “lor”.
   Response: list of all rooms

c. List all room members
   Usage: person will enter command “limeroom”.
   Response: list of all members in the room.

d. Create a room
   Usage: person will enter command “cor”.
   Response: room with given name will be created.

e. Join a room
   Usage: person will enter command “jor”.
   Response: client will join the room with a given name.

f. Leave a room
   Usage: person will enter command “eor”.
   Response: client will leave the given name room.

g. Send message in a room
   Usage: person will enter command “chatmsg”.
   Response: client will receive the confirmation message that message to given roomname is delivered.

h. Get help
   Usage: person will enter command “help”.
   Response: client will receive the text describing all the available commands.

i. Exit the running client

Usage: person will enter command “exit”

Response: person will stop the execution of the current client process and he will be prompted with the new terminal command instance.

j. Error responses

Some examples of the error response

```
INVALID_ROOM_NAME = "invalid room name"
ROOM_DOES_NOT_EXIST = "room does not exist"
NOT_MEMBER = "you are not member"
INVALID_MESSAGE_FORMAT = "invalid message format"
```

6.1   First message sent to the server

```
struct irc_pkt_hello {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_HELLO, .length = 24 };
    uint32_t ver_magic = 0xFACE0FF1;
    char chat_name[20];
};
```

MUST return the error IRC_ERR_NAME_EXISTS. The client can then attempt to reconnect with a different name.

6.12. Listing Rooms

```
struct irc_pkt_list_rooms {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_LIST_ROOMS, .length = 0 };
}
```

6.1.1 Usage

Sent by the client to Creating a list of all of the rooms currently occupied by at least one other client.

6.1.2. Response request

Server MUST return an irc_pkt_list_resp with opcode IRC_OPCODE_LIST_ROOMS_RESP with a list of the names of all currently occupied rooms.

6.3 Joining and Room creations

```
   struct irc_pkt_join_room {
       struct irc_pkt_header header =
           { .opcode = IRC_OPCODE_JOIN_ROOM, .length = 20 };
       char room_name[20];
   }
```

6.3.1. Usage

Sent by the client to join a chat room. If no room by that name exists, one is created.

Upon joining a room, the server MUST send an IRC_OPCODE_LIST_USERS_RESP message to all users currently in that room to alert them that the user list has changed. The identifier MUST be set to the name of the room, and the item_names list MUST include a list of all of the users in the room.

Every time the room's user list changes the server MUST send a new IRC_OPCODE_LIST_USERS_RESP message to all users in the room informing them of the new room membership.

6.3.2. Field Definitions

o room_name - Name of the room to join or create. MUST follow label semantics.

6.4. Leaving a Room

```
   struct irc_pkt_leave_room {
       struct irc_pkt_header header =
           { .opcode = IRC_OPCODE_LEAVE_ROOM, .length = 20 };
       char room_name[20];
   }
```

6.4.1. Usage

Sent by the client to leave a chat room.

Upon receiving this message the server MUST remove the client from the specified room and MUST send an IRC_OPCODE_LIST_USERS_RESP message to all users currently in that room to alert them that the user list has changed. The identifier MUST be set to the name of the room, and the item_names list MUST include a list of all of the users in the room.

The server SHOULD ignore leave requests when the client is not currently a member of the specified room.

6.4.2. Field Definitions

- o room_name - Name of the room to leave. MUST follow label semantics.

6.5. Sending Messages

```
struct irc_pkt_send_msg {
    struct irc_pkt_header header =
        { .opcode = IRC_OPCODE_SEND_MSG
              <OR>  IRC_OPCODE_SEND_PRIV_MSG, .length = LENGTH };
    char target_name[20];
    char msg[LENGTH - 20];
}
```

6.5.1. Usage

Sent by a client to send a text message to either a room or another user.

If the opcode is IRC_OPCODE_SEND_MSG, then the target is a room and, after validating this message, the server MUST send an IRC_OPCODE_TELL_MSG message to all users in the specified room with the target_name parameter set to the room name, and the sending_user parameter set to the name of the user who sent the message. The server MAY choose to send messages to rooms that the client is not a member of.

If the opcode is IRC_OPCODE_SEND_PRIV_MSG, then the target is another user and, after validating this message, the server MUST send an IRC_OPCODE_TELL_PRIV_MSG message to that specific user with the forwarded message and the sending_user parameter set to the name of the user who sent the message. The target_name value in the response is unused.

6.5.2. Field Definitions

- o target_name - Name of the entity to send the message to.
- o msg - Message to send to the room.

  MUST be less than 8000 characters long.

  MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).

  MUST be NULL terminated.

  MUST not contain extra NULL terminators within the payload, they may only be at the very end.

  If the message breaks any of these rules, the server MUST terminate the connection, and MAY provide the error code IRC_ERR_ILLEGAL_MESSAGE

7. Server Messages

According to the client command or error message, a server will respond. An "invalid username" message will be shown for instance, if the username is less than two characters. The server application will stop running if exit is typed in the command line terminal of the server. As an instance, this section contains all server messages.

```
SERVER_STARTED = "server started: ctl + c to exit or exit to exit"
SERVER_STOPPED = "server stopped"
NEW_CLIENT = "new client"
WELCOME_CLIENT = "welcome user \n"
CLIENT_INPUT = "client input received"
CLIENT_INVALID = "client invalid"
CLIENT_DISCONNECT = "client disconnected"
CLIENT_DISCONNECT_ERR = "client disconnect error"
CLIENT_ALREADY_IN = "you are already in the system"
CLIENT_EXISTS = "name exists"
INPUT_INVALID = "input invalid"
UNKNOWN_COMMAND = "command invalid Type - HELP"
ROOMS_AVAILABLE_TITLE = "Available rooms \n"
NO_ROOMS_TITLE = "Sorry, no rooms "
INVALID_ROOM_NAME = "invalid room name"
ROOM_DOES_NOT_EXIST = "room does not exist"
```

```
NEW_MEMBER_JOINED = "new member joined in room"
ROOM_MEMBERS = "room members \n"
ROOM_ADDED = "new room added"
ALREADY_MEMBER = "you are already member"
MEMBER_LEFT = "one member left the room"
YOU_LEFT_ROOM = "you left the room"
MEMBERSHIP_GRANTED = "Membership granted to the room"
NOT_MEMBER = "you are not member"
INVALID_MESSAGE_FORMAT = "invalid message format"
EXIT_SUCCESSFUL = "'\n exit successful"
```

7.1. Listing Response

```
   struct irc_pkt_list_resp {
       struct irc_pkt_header header =

           { .opcode = IRC_OPCODE_LIST_ROOMS_RESP
                <OR>   IRC_OPCODE_LIST_USERS_RESP, .length = LENGTH };
       char identifier[20];
       char item_names[LENGTH/20 - 1][20];
   }
```

7.1.1. Usage

Generic listing response message sent by the server to inform the client of a list. Used for both listing rooms and listing users in a room.

7.1.2. Field definitions

- identifier - Used only for IRC_OPCODE_LIST_USERS_RESP, contains the name of the room to which the users belong. MUST follow label semantics.

item_names - Array of item names, MUST follow label semantics.

- 7.2. Forwarding Messages to Clients

```
   struct irc_pkt_tell_msg {
       struct irc_pkt_header header =
           { .opcode = IRC_OPCODE_TELL_MSG
                 <OR>  IRC_OPCODE_TELL_PRIV_MSG, .length = LENGTH };
       char target_name[20];
       char sending_user[20];
       char msg[LENGTH - 40];
   }
```

7.2.1. Usage

If the opcode is IRC_OPCODE_TELL_MSG, then this is a forwarded message from the server indicating that the specified msg was posted to a room the client is joined to. Server MUST set the target_name field to the name of the room to which the message belongs. Clients SHOULD ignore this message if target_name does not match one of the rooms the client believes it is joined to.

If the opcode is IRC_OPCODE_TELL_PRIV_MSG, then this is a forwarded private message from another user intended specifically for the recipient. Server MUST set the target_name field to the name of the intended recipient. Clients MAY choose to confirm this value is correct.

7.2.2. Field Definitions

o target_name - Name of the user or room that the message was sent to.

o sending_user - Name of the user who sent the message

o msg - Message sent to the room. MUST be less than 8000 characters long.

MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).

8. 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 and MAY choose to reconnect.

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

9. "Extra" Features Supported

Note that private messaging is supported in addition to meeting the other remaining project criteria.

10. Conclusion & Future Work

The framework for generic message passing provided by this specification enables numerous clients to connect with one another via a unified forwarding server. Future developments can include the completion of private messaging, chat room, file transmission, and cloud server deployment.

11. Security Considerations

No defence exists against scrutiny, tampering, or blatant forgery for messages sent over this mechanism. All messages sent through the use of this service are not accessible to the server.

12. IANA

Considerations None

12.1. Normative References

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

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

13. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.