Portland State University
Internet-Draft
Expires: December 2021

M. O'Tousa
K. Burton
May 2021

CS594/494 IRC Class Project

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

Abstract

The goal of this document is to describe the inner workings of our IRC application which uses a client/server architecture built with Python 3 using socket and socketserver libraries for CS594/494 Internetworking Protocols class.

Copyright Notice

Table of Contents

Introduction

We have been tasked with creating a project for our CS594/494 Internetworking protocols class. For our project we have decided on implementing our own version of an IRC application. This will be implemented using Python 3 using the sockets, and socket server libraries. This implementation will be using a server/client architecture using JSON objects to communicate over the sockets. Each JSON object will contain an opcode along with a payload.

Some design decisions include no previous history of messages for each room will be held by the server, the client will have to be logged into the room in order to access the chat, joining a room that doesn't exist creates a new room.

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. [RFC 2119]

All lines between a pair of brackets '{ }' should be interpreted as a JSON object.

Message Infrastructure

Message Format

The message is a JSON-encoded object with a top level property called “op”, containing an operation code corresponding to one of the available operations. The other properties of the message body are determined by these opcodes, and if any of these corresponding properties do not actually exist on the message, or contain invalid data, a malformed error will be returned.

Operation Codes

```
ERR = 0x0

LOGIN = 0x1
LIST_ROOMS = 0x2
LIST_USERS = 0x3
JOIN_ROOM = 0x4
LEAVE_ROOM = 0x5
MESSAGE = 0x6
USER_EXIT = 0x7
HEARTBEAT = 0x8
WHISPER = 0x9
```

Error Messages

```
ERR_UNKNOWN = 0xA
ERR_ILLEGAL_OP = 0xB
ERR_ILLEGAL_LEN = 0xC
ERR_NAME_EXISTS = 0xD
ERR_ILLEGAL_NAME = 0xE
ERR_ILLEGAL_MSG = 0xF
ERR_MALFORMED = 0x10
ERR_TIMEOUT = 0x11
ERR_ILLEGAL_WISP = 0x12
ERR_NOT_IN_ROOM = 0x13
```

Heart Beat

This is a simple heartbeat operation. A heartbeat should be sent back and forth between the server and all connected clients every second. If a heartbeat has not been responded to within 5 seconds, the connection is considered to be disconnected.

```
{
  'op': HEART_BEAT,
}
```

Login

This operation allows a client to log in with a given username. If the user name is currently active, the server must reject this login attempt with an error "ERR_NAME_EXISTS" and "ERR_ILLEGAL_NAME" if the name contains characters that are not allowed by the server.

Client sends:

```
{
  'op': LOGIN,
  'username': 'user0',
}
```

Server responds with a unique identifier for the user:

```
{
  'op': LOGIN,
  'username': 'user0',
}
```

List Rooms

This operation allows a client to get a complete list of rooms available on the server. Only the client that requested the list of rooms gets a response, no other clients are notified. It is up to the server implementation to decide if clients receive all rooms, or if there is some level of access required and the server's response is potentially not every room that exists on the server.

Client sends:

```
{
  'op': LIST_ROOMS,
```

```
}
```

Server responds with:

```
{
  'op': LIST_ROOMS,
  'rooms': [
     'Default',
     'Room1',
     …
   ]
}
```

List Users

This operation allows a client to get a complete list of all rooms available on the server. Only the client that requested the list of rooms gets a response, no other clients are notified. If it matches a room name, only the users in that room are returned.

Client sends:

```
{
  'op': LIST_USERS,
  'room': 'room_name',
}
```

Server responds with:

```
{
  'op': LIST_USERS,
  'users': [
    'user_name0',
    'user_name1',
    ...
  ]
}
```

Join Room

This operation allows a client to join an already existing room or creates a new room if the room doesn't already exist. Clients should expect to see a message back saying they've joined the room. This message also contains a bool value that states if the room was newly created or if the user joined an existing room.

```
{
  'op': JOIN_ROOM,
  'user': 'username',
  'room': 'room_name'
}
```

Server responds with:

```
{
  'op': JOIN_ROOM,
  'user': 'username',
  'room': 'room_name',
  'new' : BOOL
}
```

Leave Room

This operation allows a client to leave a room it has previously joined. Clients that leave a room will have the room's message history deleted, and will have to re-join in order to send or receive communications in that room. The server notifies all clients that are in the room that the user has left. Clients that leave a room expect to see the notification from the server that they have left the room as a confirmation that leaving was successful.

Client sends:

```
{
  'op': LEAVE_ROOM,
  'user': 'username'
  'room': 'room_name'
}
```

Server responds to all clients in room with:

```
{
  'op': LEAVE_ROOM,
  'room': 'room_name',
  'user': 'username',
}
```

Message

This operation allows a client to post a message to the server, which will in turn notify all clients currently in the room the message was sent to that the message has been posted. Clients that post a message expect to see their sent message re-broadcast as an acknowledgement.

Client sends:

```
{
  'op': MESSAGE,
  'user': 'username',
  'room': 'room_name',
  'message': 'This is a message.'
}
```

Server responds to all clients with:

```
{
  'op': MESSAGE,
  'user': 'username',
  'room': 'room_name',
  'message': 'This is a message.'
}
```

Whisper

This operation allows users to send messages to specific users. It works by creating a room that's only visible to the sender and target. It's giving a room name 'sender.target'. The client checks to make sure that 'target.user' doesn't already exist.

Client sends:

```
{
  'op': WHISPER,
  'sender': 'username',
  'target': 'target_name',
  'Message': 'this is the message'
}
```

Server Sends:

```
{
  'op': WHISPER,
  'sender': 'username',
  'target': 'target_name',
  'room': 'username.target_name'
  'message': 'this is the message'
}
```

Exit_app

When the server detects a client has left it broadcasts to all clients that the user has left the server.

Server sends:

```
{
  'op': USER_EXIT,
  'user': 'username',
}
```

Error Handling

In the case of invalid or malformed messages, some error messages are sent back to the client. If the server loses connection to a client, the server will remove that client from the list of current users. If the client loses connection to the server, it will consider itself disconnected. Clients that disconnect are allowed to reconnect to the server.

Unknown

Not currently implemented. Would be used if a bad message is received. Illegal_op is currently being used instead.

Illegal Operation

Server sends:

```
{
  'op': ERR_ILLEGAL_OP,
}
```

Illegal Name

If the name is less than 1 character or greater than 32 the server sends an error message.

Server sends:

```
{
  'op': ERR_ILLEGAL_LEN
  'user': 'username'
}
```

Name Exists

If the client tries to login with a username that already exists then it receives an error message stating it already exists.

Server sends:

```
{
  'op': ERR_NAME_EXISTS
  'user': 'username'
}
```

Illegal Name

If the name contains a period '.' then the server sends back an error. This occur because the character is reserved for private messages

Server sends:

```
{
  'op': ERR_ILLEGAL_NAME
  'user': 'username'
```

```
}
```

Illegal Messages

Not currently implemented. Would be used to potentially ban messages or words.

Malformed

Not implemented. Would be used if JSON is missing a field.

Timeout

Not implemented. Socket already handles timeouts gracefully.

Illegal Whisper

If the client tries to send a whisper to itself the server sends back an error message.

Server sends:

```
{
  'op': ERR_ILLEGAL_WISP,
  'user': 'username'
}
```

Leaving Room user not in

When the client tries to leave a room which they currently are not in, the server responds with an error code. This includes rooms that don't exist, or rooms they are not in.

Server sends:

```
{
  'op': ERR_NOT_IN_ROOM
}
```

Conclusion & Future Work

This is a generic IRC protocol that allows multiple clients to communicate through a central server. Without any modification, clients and servers could be built upon this protocol to transmit other kinds of data as well, such as base64-encoded binary data.

Security Considerations

There is no security. Messages may be easily inspected or tampered with. There is no encryption, or real protection against anything.

IANA Considerations

None

Acknowledgements

Drafted using Google Docs.

Formatted and styled using the RFC style guide and the sample RFC provided in class.

<https://tools.ietf.org/id/draft-flanagan-7322bis-02.html#IAB-FORM>
Internet Relay Chat Class Project draft-irc-pdx-cs594-00.txt

Authors' Address