

Abstract

This document outlines the implementation of a server-side application structured to facilitate a robust Internet Relay Chat (IRC) environment, adhering to the specifications delineated in RFC 2812 (Internet Relay Chat: Client Protocol) and RFC 2813 (Internet Relay Chat: Server Protocol). The server is designed to handle multiple client connections, manage user and channel operations, and support server-to-server interactions, thereby constituting a comprehensive IRC network system.

Implementation Details

The server application is built using C programming language, leveraging standard networking and threading libraries provided by UNIX/Linux systems, such as `sys/socket.h`, `arpa/inet.h`, and `pthread.h`.

This setup enables the server to handle asynchronous network communication effectively, manage multiple user sessions concurrently, and execute IRC commands as per the protocols.

Setting up the server

- **Compilation:** The server is written in C and can be compiled using a standard C compiler like gcc. The command to compile might look like `gcc -o irc_server server.c -lpthread`, where `irc_server` is the output executable, and `server.c` is the source file. The `-lpthread` flag links the POSIX threads library, essential for handling multiple client connections.
- **Running the Server:** After compilation, the server is started by executing `./irc_server`. It listens on the predefined port (e.g., 8080) for client connections.

Handling client connections

- The server uses TCP sockets to accept incoming connections. It listens on a specified port and waits for clients to connect.
- Upon a client's connection request, the server accepts the connection, creating a new socket specifically for communicating with that client.
- Each client connection is handled in a separate thread, allowing the server to manage multiple clients concurrently without blocking.

Command processing

- The server reads the incoming data stream from each client and parses the data as commands according to the IRC protocol.
- Common commands include:
 - **NICK:** Sets or changes a user's nickname.
 - **USER:** Used at the beginning of a connection to set the user's mode, username, and real name.
 - **JOIN:** Joins a channel or creates it if it doesn't exist.

- PART: Leaves a channel.
- PRIVMSG: Sends a private message to a user or a channel.
- TOPIC: Sets or gets the topic of a channel.
- NAMES: Lists all nicknames visible to the user (in a channel or the entire server).
- TIME: Returns the local time of the server.
- Commands are case-insensitive and parsed against a predefined set of rules to ensure compliance with the IRC standards.

Channel management

- Channels are created dynamically when the first user joins a non-existing channel with the JOIN command.
- Each channel has a list of clients and maintains information such as channel name, topic, and participant list.
- The server handles the channel leaving with the PART command, removing the user from the channel's client list. If the last user leaves, the channel is effectively deleted.

Message handling

- For PRIVMSG commands, the server checks whether the target is a user or a channel.
- If it's a user, the server finds the client socket associated with the target nickname and forwards the message to that socket.
- If the target is a channel, the server broadcasts the message to all sockets connected to that channel, except the sender.
- TOPIC command allows users to set or query the topic of a channel.
- The NAMES command sends a list of all users in a specified channel, or all channels if no channel is specified.
- TIME command sends the local time of the server to the requesting user.

This comprehensive handling of connections, commands, and messages enables the server to function as a basic but fully operational IRC server.

Description of Base64 Encoding in Client.c:

In the provided IRC client implementation, Base64 encoding plays a crucial role in enhancing the security and integrity of communication, particularly in the transmission of sensitive data such as passwords. The `base64_utils.h` library is employed to encode passwords before they are sent from the client to the IRC server. This encoding approach is instrumental in ensuring that passwords are not transmitted in plaintext over the network, which helps to mitigate risks associated with eavesdropping or man-in-the-middle attacks.

Commands Screenshots :

JOIN :

```
JOIN #foo  
Enter message (type 'QUIT' to exit):  
Server response: Channel #foo  
created and client 5 joined.
```

```
JOIN #foo  
Enter message (type 'QUIT' to exit):  
Server response: Client 4 joined channel #foo
```

NICK:

```
Enter message (type 'QUIT' to exit):  
NICK bro1  
Server response: :128.226.114.201 401 bro1  
bro1  
:Nickname is now bro1
```

```
Enter message (type 'QUIT' to exit):  
NICK bro2  
Server response: :128.226.114.201 401 bro2  
bro2  
:Nickname is now bro2
```

TOPIC:

```
Enter message (type 'QUIT' to exit):  
TOPIC #foo :Networking  
Server response: Topic for #foo set to :Networking  
  
Enter message (type 'QUIT' to exit):  
TOPIC #foo :  
Server response: Topic for #foo set to :
```

TIME:

```
Enter message (type 'QUIT' to exit):  
TIME  
Server response: :1711722570 391 :Local time is 10:29:30
```

USER :

```
USER try1 :Shubham Navale  
Enter message (type 'QUIT' to exit):  
Server response: :bar.example.com 001 try1  
:Welcome to the IRC network try1  
!try1  
@bar.example.com
```

NAMES:

```
NAMES #foo  
Enter message (type 'QUIT' to exit):  
Server response: Channel: #foo  
  
User: try2  
  
User: try1
```

PASS :

```
numparams val : 2Password received and stored for client 4  
Password for socket 4 saved successfully in .usr_pass file.
```

```
pveera1@remote04:~/PRJ02$ ./client client.conf  
PASS trypass1
```

.usr_pass :

```
4:dHJ5cGFzcZE=
```

PRIVMSG:

```
PRIVMSG try2 Hii
```

```
Server response: From : Hii
```

PART:

```
PART #foo  
Enter message (type 'QUIT' to exit):  
Server response: Client 5 left channel #foo
```

```
PART #foo  
Enter message (type 'QUIT' to exit):  
Server response: Client 4 left channel #foo
```

QUIT:

```
QUIT  
pveera1@remote03:~$
```

Testing Commands :

To thoroughly test the functionalities of your IRC-like server, you can use the following series of test

commands. These commands should cover the primary functionalities such as joining a channel, parting from a channel, setting or getting a topic, listing names, and sending private messages.

1. Connecting to the Server

- Client connects to the server (no command needed, just establishing connection)

2. Setting a Nickname

- `NICK username :password`

3. Registering as a User

- `USER username * * :Real Name`

4. Joining a Channel

- `JOIN channel1`
- `JOIN channel2`

5. Parting from a Channel

- `PART channel1`

6. Setting a Topic for a Channel

- `TOPIC channel2 :This is a new topic for channel2`

7. Getting the Topic for a Channel

- `TOPIC #channel2`

8. Listing All Names in a Channel

- `NAMES channel2`
- `NAMES` (to list all channels and names if implemented)

9. Sending a Private Message to a User

- `PRIVMSG username :Hello, how are you?`

10. Sending a Message to a Channel

- `PRIVMSG #channel2 :Hello everyone in channel2`

11. Querying Server Time

- `TIME`

12. Quitting the Server

- `QUIT`

Example Interaction for Testing:

- Client 1 connects and sets their nickname and user:

...

NICK Alice

USER Alice :Alice Smith

...

- Client 1 joins a channel:

...

JOIN #testChannel

...

- Client 2 connects and sets their nickname and user, then joins the same channel:

...

NICK Bob

USER Bob :Bob Jones

JOIN #testChannel

...

- Client 1 sets a topic for the channel:

...

TOPIC #testChannel :Welcome to Test Channel

...

- Client 2 gets the topic for the channel:

...

TOPIC #testChannel

...

- Client 1 lists all names in the channel:

...

NAMES #testChannel

...

- Client 2 sends a private message to Client 1:

...

PRIVMSG Alice :Hi Alice!

...

- Client 1 sends a message to the channel:

...

PRIVMSG #testChannel :Hello everyone here!

...

- Client 1 parts from the channel:

...

PART #testChannel

...

- Client 2 queries the server time (if implemented):

...

TIME

...

- Client 1 quits:

...

QUIT

...

- **Appendices**

IRC Documentations

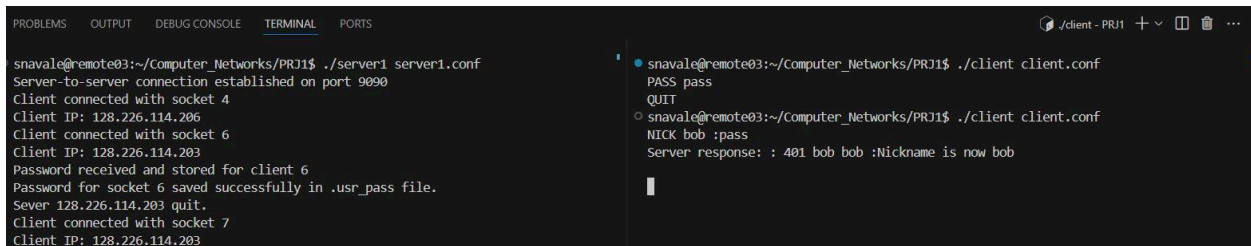
- IRC (Internet Relay Chat) provides a way for multiple users to communicate in real-time. The protocol facilitates communication in the form of text messages, allowing conversations in private or in channels (group chat).

RFC 2812

- RFC 2812 is a document that specifies the technical details of the IRC protocol. It defines the architecture, core concepts, commands, and message formats used in IRC. Key aspects covered include:
 - Client and Server Architecture: Describes how clients connect to servers and interact within the IRC network.
 - Commands and Responses: Details the commands used for communication, such as NICK, JOIN, PART, PRIVMSG, MODE, and the responses or numeric replies generated by the server.
 - Channels and Modes: Explains how channels are created, managed, and the different modes that can be applied to users and channels for controlling behavior and access.
 - Messaging and Communication Protocols: Outlines the protocols for sending private and channel-wide messages, including message formatting and character encoding standards.

The adherence to RFC 2812 ensures compatibility and standardization across different IRC implementations, facilitating a uniform experience for users across various IRC networks.

server -client password authentication:



The screenshot shows a terminal window with two panes. The left pane displays the output of a server program, and the right pane shows the input and output of a client program.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
snavale@remote03:~/Computer_Networks/PRJ1$ ./server1 server1.conf
Server-to-server connection established on port 9090
Client connected with socket 4
Client IP: 128.226.114.206
Client connected with socket 6
Client IP: 128.226.114.203
Password received and stored for client 6
Password for socket 6 saved successfully in .usr_pass file.
Sever 128.226.114.203 quit.
Client connected with socket 7
Client IP: 128.226.114.203

snavale@remote03:~/Computer_Networks/PRJ1$ ./client client.conf
PASS pass
QUIT
snavale@remote03:~/Computer_Networks/PRJ1$ ./client client.conf
NICK bob :pass
Server response: : 401 bob bob :Nickname is now bob
```

Error codes Handled:

Error 101: Need more parameters;

Error 102: Too many parameters;

Error 103: Channel is full;

Error 104: No such channel;

Error 105: Too many channels;

Error 106: Too many targets;

Error 107: Unavailable resource";

Error 201: Not on channel;

Error 301: Too many matches;

Error 302: No such server;

411 :No recipient given (PRIVMSG)\n";

412 :No text to send\n;

401 :No such nick/channel\n;

error_message = No topic;

error_message = Unknown error;

SERVER TO SERVER commands :

PASS:

```
snavale@remote03:~/Computer_Networks/PRJ1$ ./server1 server1.conf
Server-to-server connection established on port 9090
Client connected with socket 4
Client IP: 128.226.114.206
PASS supersecretpassword 0210010000 IRC|aBgH$
```

```
Server-to-server connection established on port 8080
Version: 0210010000
Flags: IRC|aBgH$
Password verified for returning client/server IP 128.226.114.203.
Password set/updated for client IP 128.226.114.203.
```

NICK: NICK <nickname>

```
NICK babbage2
```

```
Server response: : 401 babbage2 babbage2 :Nickname is now babbag
```

NJOIN : NJOIN <channel>

```
snavale@remote03:~/Computer_Networks/PRJ1$ ./server1 server1.conf
Server-to-server connection established on port 9090
Client connected with socket 4
Client IP: 128.226.114.206
NICK babbage1
Server response: : 401 babbage1 babbage1 :Nickname is now babbag
NJOIN #channel1
```

PRIVMSG: PRIVMSG <servernick> <clientnick> <message>

```
snavale@remote03:~/Computer_Networks/PRJ1$ ./server1 server1.conf
Server-to-server connection established on port 9090
Client connected with socket 4
Client IP: 128.226.114.206
NICK babbage1
Server response: : 401 babbage1 babbage1 :Nickname is now babbag
Server 4 set nickname to babbage2
Client connected with socket 6
Client IP: 128.226.114.203
Password received and stored for client 6
Password for socket 6 saved successfully in .usr_pass file.
Client 6 set nickname to bob
```

```
snavale@remote03:~/Computer_Networks/PRJ1$ ./client client.conf
PASS pass
NICK bob :pass
Server response: : 401 bob bob :Nickname is now bob
PRIVMSG babbage2 alice :Hello
```

```
pveera1@remote04:~/PRJ02$ ./client client.conf
PASS computer
NICK alice :computer
Server response: : 401 alice alice :Nickname is now alice
Server response: :Hello
```

SQUIT : SQUIT

```
Server-to-server connection established on port 8080
SQUIT
Server response: BYE
Server response: ELF
libgcc_s.so.1 must be installed for pthread_exit to work
Aborted
```

Conclusion:

The server's architecture is aligned with the IRC standards, providing a functional and expandable framework for real-time text communication. Future enhancements could include advanced authentication mechanisms, increased scalability features, and extended support for IRC extensions. The implementation serves as a solid foundation for building a comprehensive IRC network capable of supporting varied user interactions in real-time.