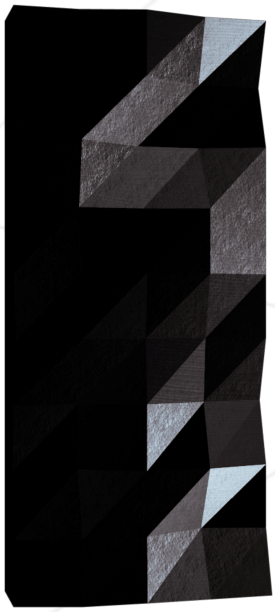

ft_irc

Internet Relay Chat

Summary:
This project is about creating your own IRC server.
You will use an actual IRC client to connect to your server and test it.
Internet is ruled by solid standards protocols that allow connected computers to interact with each other.
It's always a good thing to know.

Version: 7

Contents

Chapter I

Introduction

Internet Relay Chat or IRC is a text-based communication protocol on the Internet. It offers real-time messaging that can be either public or private. Users can exchange direct messages and join group channels.

IRC clients connect to IRC servers in order to join channels. IRC servers are connected together to form a network.

Chapter II

General rules

- Your program should not crash in any circumstances (even when it runs out of memory), and should not quit unexpectedly.
  If it happens, your project will be considered non-functional and your grade will be `0`.
- You have to turn in a `Makefile` which will compile your source files. It must not relink.
- Your `Makefile` must at least contain the rules:
  `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- Compile your code with `c++` and the flags `-Wall -Wextra -Werror`
- Your code must comply with the **C++ 98 standard**. Then, it should still compile if you add the flag `-std=c++98`.
- Try to always develop using the most `C++` features you can (for example, choose `<cstring>` over `<string.h>`). You are allowed to use `C` functions, but always prefer their `C++` versions if possible.
- Any external library and `Boost` libraries are forbidden.

Chapter III

Mandatory Part

Program name	`ircserv`
Turn in files	`Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp,` an optional configuration file
Makefile	`NAME, all, clean, fclean, re`
Arguments	`port:` The listening port `password:` The connection password
External functs.	Everything in C++ 98. `socket, close, setsockopt, getsockname, getprotobyname, gethostbyname, getaddrinfo, freeaddrinfo, bind, connect, listen, accept, htons, htonl, ntohs, ntohl, inet_addr, inet_ntoa, send, recv, signal, lseek, fstat, fcntl, poll` (or equivalent)
Libft authorized	n/a
Description	An IRC server in C++ 98

You have to develop an IRC server in C++ 98.

You **mustn't** develop a client.
You **mustn't** handle server-to-server communication.

Your executable will be run as follows:

```
./ircserv <port> <password>
```

- `port`: The port number on which your IRC server will be listening to for incoming IRC connections.
- `password`: The connection password. It will be needed by any IRC client that tries to connect to your server.

Even if `poll()` is mentionned in the subject and the evaluation scale, you can use any equivalent such as `select()`, `kqueue()`, or `epoll()`.

III.1 Requirements

- The server must be capable of handling multiple clients at the same time and never hang.
- Forking is not allowed. All I/O operations must be **non-blocking**.
- Only **1** `poll()` (or equivalent) can be used for handling all these operations (read, write, but also listen, and so forth).

`Because you have to use non-blocking file descriptors, it is possible to use read/recv or write/send functions with no poll() (or equivalent), and your server wouldn't be blocking.`
`But it would consume more system resources.`
`Thus, if you try to read/recv or write/send in any file descriptor without using poll() (or equivalent), your grade will be 0.`

- Several IRC clients exist. You have to choose one of them as a **reference**. Your reference client will be used during the evaluation process.
- Your reference client must be able to connect to your server without encountering any error.
- Communication between client and server has to be done via `TCP/IP` (v4 or v6).
- Using your reference client with your server must be similar to using it with any official IRC server. However, you only have to implement the following features:
  - You must be able to authenticate, set a nickname, a username, join a channel, send and receive private messages using your reference client.
  - All the messages sent from one client to a channel have to be forwarded to every other client that joined the channel.
  - You must have `operators` and regular users.
  - Then, you have to implement the commands that are specific to `channel operators`:
    - KICK - Eject a client from the channel
    - INVITE - Invite a client to a channel
    - TOPIC - Change or view the channel topic
    - MODE - Change the channel's mode:
      - i: Set/remove Invite-only channel
      - t: Set/remove the restrictions of the TOPIC command to channel operators
      - k: Set/remove the channel key (password)
      - o: Give/take channel operator privilege

- l: Set/remove the user limit to channel

- Of course, you are expected to write a clean code.

III.2 For MacOS only

Since MacOS doesn't implement write() the same way as other Unix OSes, you are allowed to use fcntl().
You must use file descriptors in non-blocking mode in order to get a behavior similar to the one of other Unix OSes.

However, you are allowed to use fcntl() only as follows:
fcntl(fd, F_SETFL, O_NONBLOCK);
Any other flag is forbidden.

III.3 Test example

Verify absolutely every possible error and issue (receiving partial data, low bandwidth, and so forth).

To ensure that your server correctly processes everything that you send to it, the following simple test using `nc` can be done:

```
\$> nc 127.0.0.1 6667
com^Dman^Dd
\$>
```

Use `ctrl+D` to send the command in several parts: '`com`', then '`man`', then '`d`\n'.

In order to process a command, you have to first aggregate the received packets in order to rebuild it.

Chapter IV

Bonus part

Here are the extra features you can add to your IRC server so it looks even more like and actual IRC server:

- Handle file transfer.
- A bot.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter V

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

You are encouraged to create test programs for your project even though they **won't be submitted and won't be graded**. Those tests could be especially useful to test your server during defense, but also your peer's if you have to evaluate another `ft_irc` one day. Indeed, you are free to use whatever tests you need during the evaluation process.

`Your reference client will be used during the evaluation process.`