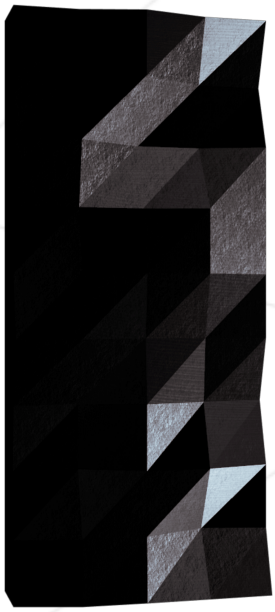

IRC

Internet Relay Chat

Summary: The goal of this project is to make you write your own IRC server. To do so, you will follow the real IRC RFC and test your work with real IRC clients. Internet is ruled by solid and standards protocols that allow a strong interaction between every connected computer. It's always good to know about it.

Contents

Chapter I

Common Instructions

- Your program should not crash in any circumstances (even when it runs out of memory), and should not quit unexpectedly. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output. Your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses for your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, librairies or functions that could be forbidden on the main part of the project. Bonuses must be in different files, clearly identified from their names. Mandatory and bonus part evaluation is done separately.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter II

Introduction

Internet Relay Chat or IRC is a textual communication protocol on the Internet. It is instantaneous communication mainly in the form of discussions in groups via discussion channels, but can also be used for one-to-one communication.

IRC client programs connect to an IRC server to access specific channel. IRC servers are connected between them to provide a global network with unique channels.

Chapter III

Mandatory Part

Program name	`ircserv`
Turn in files	
Makefile	`Yes`
Arguments	
External functs.	`socket, open, close, setsockopt, getsockname, getprotobyname, gethostbyname, getaddrinfo, freeaddrinfo, bind, connect, listen, accept, htons, htonl, ntohs, ntohl, inet_addr, inet_ntoa, send, recv, exit, signal, lseek, fstat, read, write, fcntl, select, FD_CLR, FD_COPY, FD_ISSET, FD_SET, FD_ZERO`
Libft authorized	
Description	`Write an IRC server in C++`

- Your must write an IRC server in C++
- The reference C++ version is C++98
- Follow the RFC 1459, 2810, 2811, 2812, 2813 and 7194
- Communication between client and server must be done via TCP/IP(v4) or (v6)
- You `won't need` to code a client
- You `need` to handle server to server communication
- You can include and use everything in "iostream" "string" "vector" "list" "queue" "stack" "map" "algorithm"
- You can use a cryptographic library to handle TLS communications
- For bonuses, you are allowed to use any other function, as long as its usage is dully justified during defence. Be smart.
- Your executable will be used as follows:

```
./ircserv [host:port_network:password_network] <port> <password>
```

  - `host` is the hostname on which IRC must connect to join a already existing network
  - `port_network` is the server port on which IRC must connect on `host`
  - `password_network` is the password needed to connect on `host`
  - `port` is the port number on which your server will accept incoming connections. Add one for TLS port.
  - `password` is the password needed by any IRC client or server who wants to connect to your server.
  - If `host`, `port_network` and `password_network` aren't given, you must create a `new IRC network`

- The server must be capable of handling multiple clients at the same time, along with the possible server, and never hang. Forking is not allowed, all IO operations must be non blocking and use only 1 select for all (read, write, but also listen, ...)

We've let you use fcntl because MacOS X doesn't implement write the same way as other Unix OSes.
You must use non-blocking FD to have a result similar to other OSes.

Because you are using non-blocking FD, you could use read/recv or write/send functions without select and your server would be not blocking. But it would consume more system resources.
Again trying to read/recv or write/send in any FD without going through a select will give you a mark equal to 0 and the end of the evaluation.

You can only use fcntl as follow: `fcntl(fd, F_SETFL, O_NONBLOCK);`
Any other flags is forbidden.

- You are of course expected to build a clean code. Verify absolutely every error and in cases where you might have a problem (partial data received, low bandwith...)
- To verify that your server correctly uses everything you send, an initial test can be done with `nc` (Use ctrl+d to send parts of the command):

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

This will allow you to first send the letters `com`, `man`, `d`\n. You must first agregate the packets to rebuild the command `command` in order to handle it.

- Several IRC clients exist, can be used for several tests, and will be used during the defense.
- For server to server communication, you will also need to interact with a public irc server during the defense.

Chapter IV

Bonus part

Here are the bonuses you can add to your IRC to make it closer to the actual IRC. Of course, if the mandatory part is not perfect, don't even think about bonuses.

- Client
- A configuration file, with according command line parameters modification
- Non mandatory cases of the RFC
- Graphic interface
- File transfer
- A bot
- A fully operational connection with a public IRC network.