IRC

Internet Relay Chat

Summary: The objective of this project is to reproduce the functioning of the server part of an IRC by following an RFC

Contents

Chapter I

Common Instructions

- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- All heap allocated memory space must be properly freed when necessary. No leaks will be tolerated.
- If the subject requires it, you must submit a `Makefile` which will compile your source files to the required output with the flags `-Wall`, `-Wextra` and `-Werror`, and your Makefile must not relink.
- Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To turn in bonuses to your project, you must include a rule `bonus` to your Makefile, which will add all the various headers, librairies or functions that are forbidden on the main part of the project. Bonuses must be in a different file `_bonus.{c/h}`. Mandatory and bonus part evaluation is done separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` in a `libft` folder with its associated Makefile. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- 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.

Chapter III

Mandatory Part

Your IRC must:

- Your code must be in C++
- Follow the RFC 2810, 2811, 2812, and 2813 protocols
- Communication between client and server must be done via TCP/IP(v4) or (v6)
- You `won't need` to handle client side
- You `won't need` to handle client to server authentification
- You `need to` handle server to server communication
- To complete the mandatory part, you are allowed to use the following functions:
    - socket(2), open(2), close(2), setsockopt(2), getsockname(2)
    - getprotobyname(3), gethostbyname(3), getaddrinfo(3)
    - bind(2), connect(2), listen(2), accept(2)
    - htons(3), htonl(3), ntohs(3), ntohl(3)
    - inet_addr(3), inet_ntoa(3)
    - send(2), recv(2)
    - exit(3), signal(3)
    - lseek(2), fstat(2)
    - read(2), write(2)
    - select(2), FD_CLR, FD_COPY, FD_ISSET, FD_SET, FD_ZERO
    - And any other functions used in bircd.tar.gz which wouldn't be included in this list
    - You can include and use everything in "iostream" "string" "vector" "list" "queue" "stack" "map" "algorithm"
- For bonuses, you are allowed to use any other function, as long as its usage is dully justified during defence. Be smart.

Chapter IV

Serveur

- Your executable will be named and used as follows:

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

  - `host` is the hostname on which IRC must connect
  - `port_network` is the server port on which IRC must connect on `hostname`
  - `port` is the port number for the server
  - If `host` and `port_network` aren't given, you must create a `new IRC network`

- Dont handle the special cases `5.7` and `5.8` of RFC 2813
- The server must be capable of handling multiple clients at the sane time, using a single `select(2)` for the server
- `select(2)` must be `non-blocking`.
- You are 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.

`An example server is included (bircd.tar.gz). You can use it and turn it in, but be wary: this base won't bring you any points, and the select is read-only, so you'll have to fix it. By the way it's pure c....`

Chapter V

Bonus part

Here are the bonuses you can add to your IRC to make it closer to the actual IRC.

- Client
- Cases that we asked you not to handle
- Graphic interface
- File transfer
- a bot