

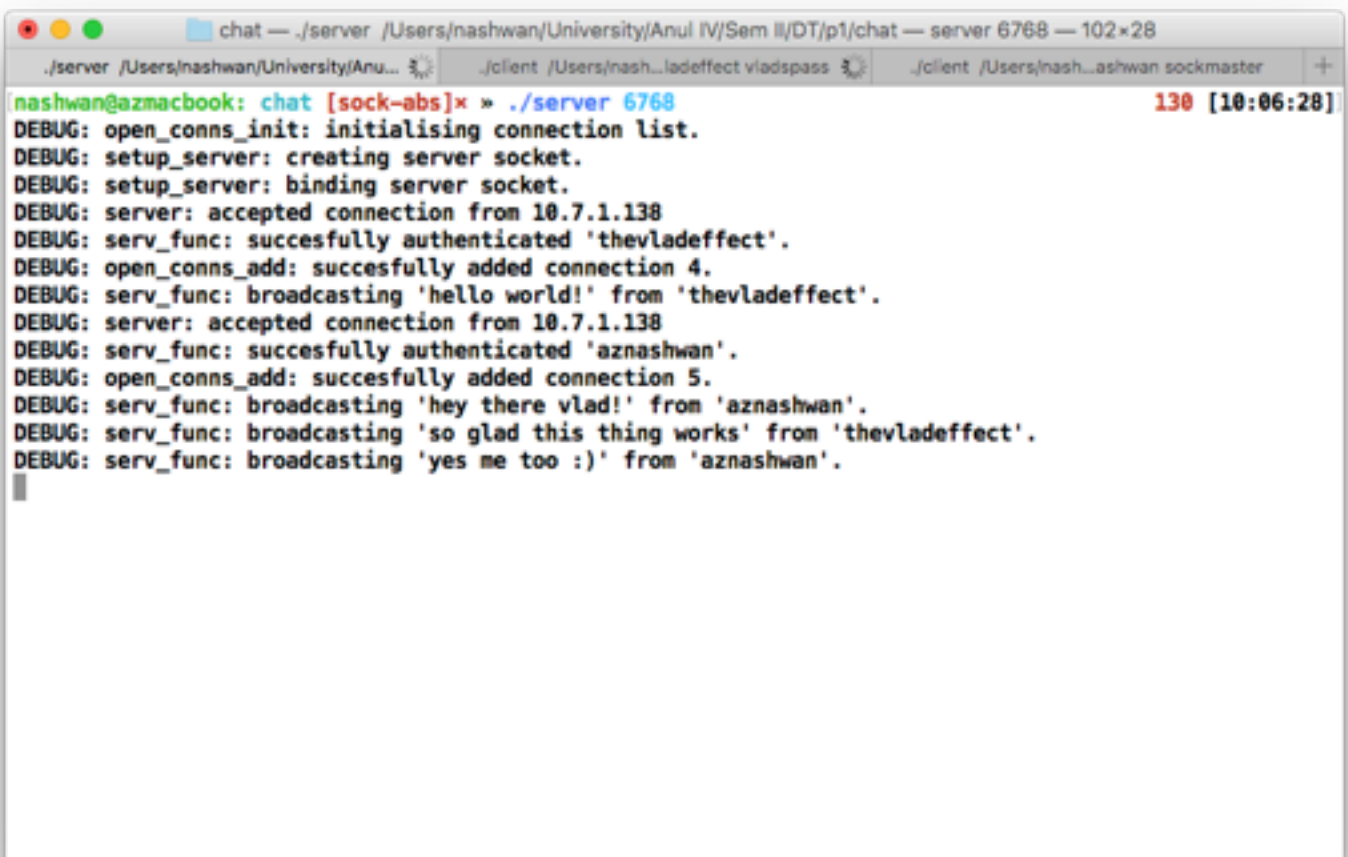
---

# CIRC

## the lightweight chatternative

Azhari, Todorovici, Vidac - March 12, 2016

---



```
nashwan@azmacbook: chat [sock-abs]x » ./server 6768
DEBUG: open_conns_init: initialising connection list.
DEBUG: setup_server: creating server socket.
DEBUG: setup_server: binding server socket.
DEBUG: server: accepted connection from 10.7.1.138
DEBUG: serv_func: succesfully authenticated 'thevladeffect'.
DEBUG: open_conns_add: succesfully added connection 4.
DEBUG: serv_func: broadcasting 'hello world!' from 'thevladeffect'.
DEBUG: server: accepted connection from 10.7.1.138
DEBUG: serv_func: succesfully authenticated 'aznashwan'.
DEBUG: open_conns_add: succesfully added connection 5.
DEBUG: serv_func: broadcasting 'hey there vlad!' from 'aznashwan'.
DEBUG: serv_func: broadcasting 'so glad this thing works' from 'thevladeffect'.
DEBUG: serv_func: broadcasting 'yes me too :)' from 'aznashwan'.
```

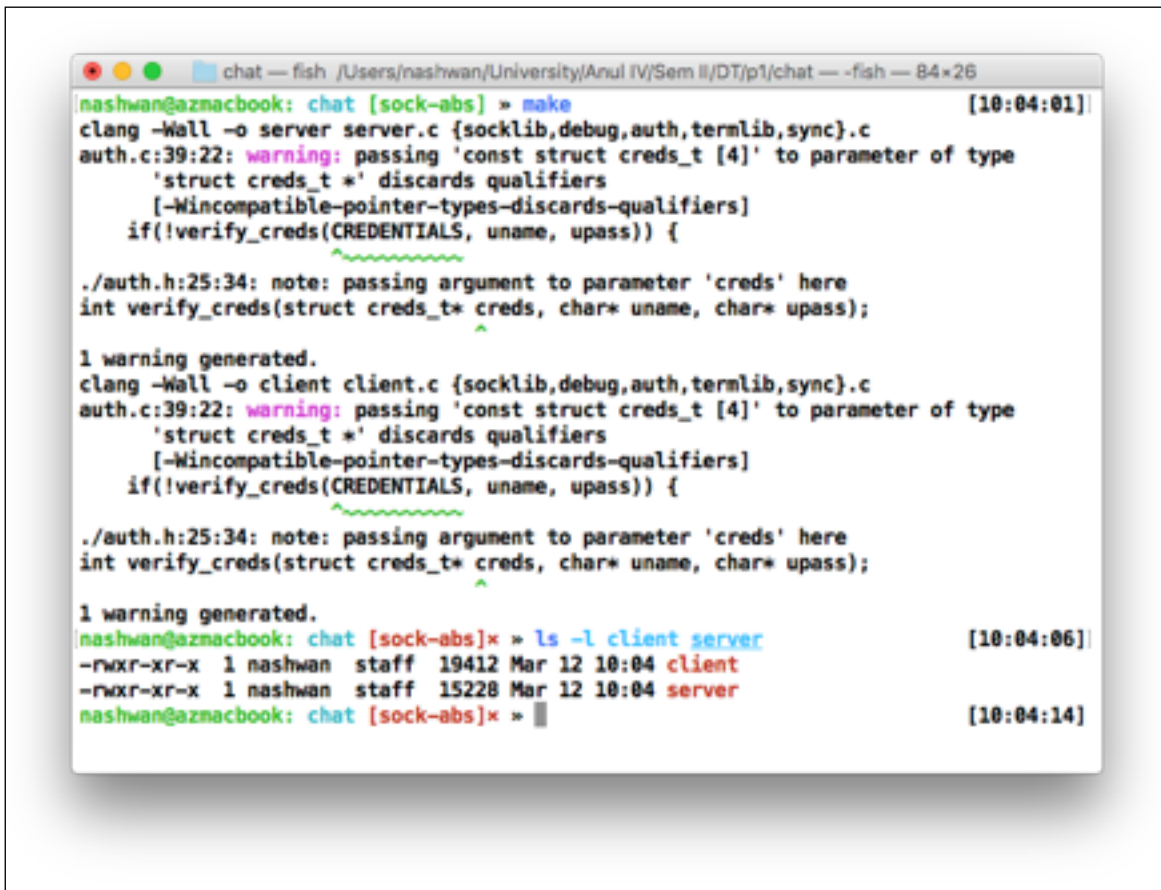
---

## Introduction

Internet chatting has a very important role to play in the development of software ever since the beginning, empowering developers everywhere to reach out to each-other and share everything from knowledge to a good laugh.

C.I.R.C. is a simple communication solution which offers unmatched ease of use and setup on a single multi-tenant channel where everyone can speak their minds.

## Setup and Installation



```
chat — fish /Users/nashwan/University/Anul IV/Sem II/DT/p1/chat — -fish — 84x26
nashwan@azmacbook: chat [sock-abs] » make [10:04:01]
clang -Wall -o server server.c {socklib,debug,auth,termLib,sysnc}.c
auth.c:39:22: warning: passing 'const struct creds_t [4]' to parameter of type
      'struct creds_t *' discards qualifiers
      [-Wincompatible-pointer-types-discards-qualifiers]
      if(!verify_creds(CREDENTIALS, uname, upass)) {
          ~~~~~
./auth.h:25:34: note: passing argument to parameter 'creds' here
int verify_creds(struct creds_t* creds, char* uname, char* upass);
          ^
1 warning generated.
clang -Wall -o client client.c {socklib,debug,auth,termLib,sysnc}.c
auth.c:39:22: warning: passing 'const struct creds_t [4]' to parameter of type
      'struct creds_t *' discards qualifiers
      [-Wincompatible-pointer-types-discards-qualifiers]
      if(!verify_creds(CREDENTIALS, uname, upass)) {
          ~~~~~
./auth.h:25:34: note: passing argument to parameter 'creds' here
int verify_creds(struct creds_t* creds, char* uname, char* upass);
          ^
1 warning generated.
nashwan@azmacbook: chat [sock-abs] » ls -l client server [10:04:06]
-rwxr-xr-x 1 nashwan staff 19412 Mar 12 10:04 client
-rwxr-xr-x 1 nashwan staff 15228 Mar 12 10:04 server
nashwan@azmacbook: chat [sock-abs] » [10:04:14]
```

C.I.R.C is written in C and must be built from source. Luckily in realistically depends only that its compiling environment be POSIX-compliant and have access to the POSIX threads API. Alternatively, one may find the use of automated build tools such as `make` or `cmake` handy. To build the project, which consists of two independent elements, a client and a server, choosing to either `make server` or `make client` should work without a hitch.

---

## Design

The project features a few important yet distinct modules:

- `debug.{h,c}`: contains functions useful for debugging, including a `stderr` logging type function and some utilities to translate IP addresses from network to string form.
- `socklib.{h,c}`: contains powerful abstractions over the standard UNIX socket API, offering a clean and easy interface to use for reading and writing messages to/ fro the listening sockets, defining socket address parameters and making it easy to create a concurrent thread-based server via a single function.
- `testing/echo/echo{server,client}.c`: simple examples showcasing `socklib`'s power.
- `auth.{h,c}`: further builds upon the interface of `socklib` to offer credential checking functionalities and a clean way of authenticating an inbound connection made to the server.
- `testing/auth/auth{client,server}.c`: showcase of how easy to integrate authentication is above the primitives of `socklib`.
- `sync.{h,c}`: set of datastructures and functions to be used for managing the open connection of a the `socklib` server, meant to run concurrently.
- `termLib.{h,c}`: simple methods which take advantage of terminal emulator's control sequences to provide a dynamic, UI-e feel to the chat client.
- `server.c`: the full implementation of the `charserver`, containing a main where `socklib::server` is initialised to handle authentication, message filtering and forwarding to all active connections in a concurrent manner.
- `client.c`: the client of the chat system, which can connect to the server given its IP, open port number, as well as the user's credentials.

## Protocol

The resolution protocol used between client and server are as simple as possible to facilitate design and reusability of resources. It relies on a set of primitives defined in `socklib` to handle all transmissions in an orderly manner based on an pre-defined `TERMINATION_CHAR`, which is meant not only to signal the end of a message, but also as a separation character between individual messages. Each message will have a predefined, pre-authenticated used behind it, after which any other coders will get the message pushed to them as well from the server.