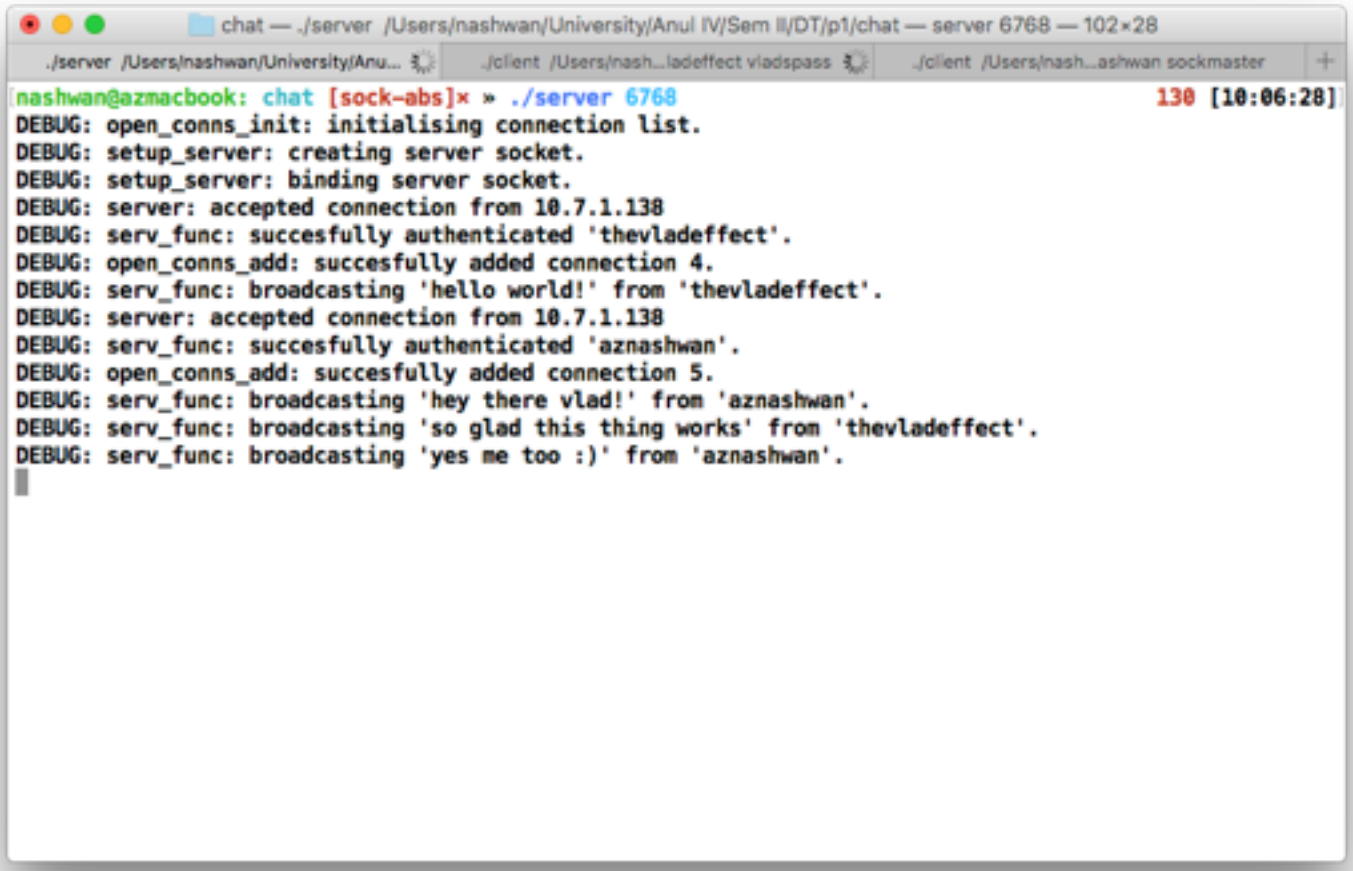
CIRC

the lightweight chatternative

Azhari, Todorovici, Vidac - March 12, 2016

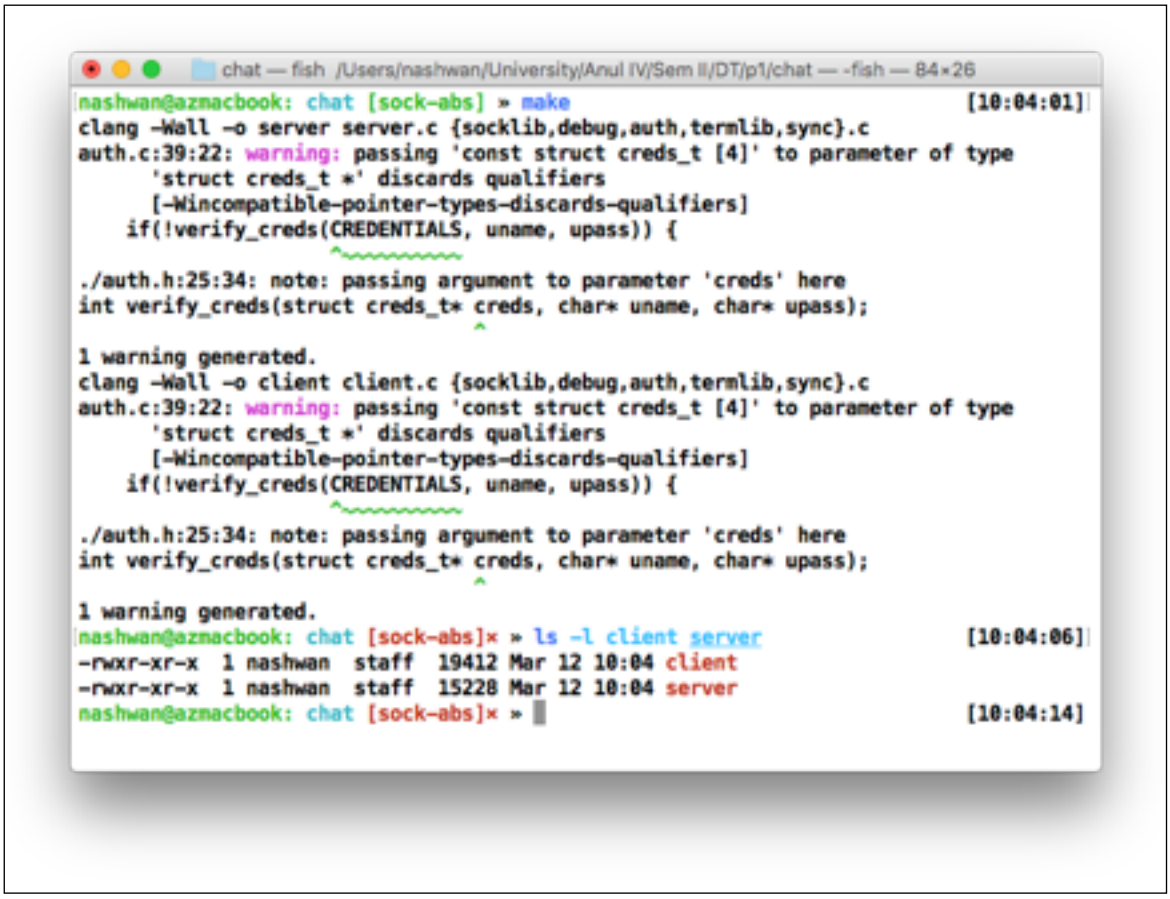


Introduction

Internet chatting has 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



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 a create 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, mean 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.