Lisp in Summer Projects Submission

Submission Date	2013-10-21 14:38:16
Full Name	Cameron Matheson
Country	USA
Project Name	B-ouncé
Type of software	service
General category	other
LISP dialect	Clojure
GitHub URL	https://github.com/cmatheson/irc-proxy
Did you start this project?	Yes, all the code is written by me
Project Description	I want to describe my project in this form.
Purpose	B-ouncé is a new IRC Proxy/Bouncer. It lets you connect to an IRC server with several clients at once (e.g., you could connect from your phone/laptop/desktop using the same nick (chat name).
Function	B-ouncé runs on a single machine and acts as a proxy. B-ouncé acts as an IRC Client to the server you really want connect to (e.g., freenode). Clients connect to the proxy (which pretends to be freenode).
Motivation	I've used other Bouncers in the past (like bip), but I'm not happy with the way they work, particularly in respect to backlog replay, (which, unfortunately, I didn't have time to address).
Audience	Anyone that uses IRC would benefit from an IRC proxy. B-ouncé is currently geared towards people that want to connect to a network with multiple devices simultaenousl.
Methodology	This was my first Clojure project, so I think I made some unfortunate mistakes along the way. I chose Clojure right after core.async was announced. The conceptual model I

	was going for was a single-threaded process that uses channels to coordinate event handling between the clients/server. Java's normal IO stuff is blocking, however, so I had to use core.async's blocking channel operations/threads for this project. Had I been more familiar with the Java/Clojure ecosystems, I probably would have used ztellman's aleph stuff (which is similar to core.async but wraps NIO). B-ouncé works by connecting to the server and listens for clients. Most messages from the server are relayed directly to all clients, although some need special handling due to the nature of the IRC protocol. To facilitate special handling of IRC messages, I wrote a simple IRC message parser. The parser just returns maps, so it would be useful outside of the context of B-ouncé as well.
Conclusion	I learned a lot about clojure and CSP doing this project. The learning curve was steep (in particular, I read signifant portions of Hoare's CSP book, but it turns out that looking at Go code is more practical for learning about practical CSP program design). I didn't have time to implement the feature I was most interested in (fancy backlog-replay (which is just a way of playing back previous messages so you can see what happened while you were away, search through previous conversations, etc.). Future plans: * implement backlog replay * allow connecting to multiple IRC networks at once * allow multiple users in the same process * switch to clojurescript (or use Netty?) so I can use core.async's non-blocking primitives. (or possibly even look into some of the new reactive stuff that is coming out).
Test Instructions	Replace the IRC server in config.edn with the information you desire (or just use the default freenode stuff). Run the server on a machine (lein run). Connect to the server with irc client(s). E.g.: irssi -c localhost Enjoy using IRC from multiple clients at the same time.
Execution Instructions	lein run
Describe any bugs or caveats	There is a subtle race condition that can be encountered due to my use of threads (when my mental model was expecting single-threaded stuff). It is rare but you may end up having multiple messages mixed together for the clients.
Official	I have read rules and have abided by them. I am 18 years of age or older. I am not living in Brazil, Quebec, Saudi Arabia, Cuba, Iran, Myanmar (Burma), North Korea, Sudan, or Syria.