Neil Resnik & Yifei Wang

Networking Overview

Introduction

This is an implementation of a Server - Client architecture with non-blocking sockets and a multi-threaded server. The server accepts connections, and spins off new threads to handle each connection in turn. The clients simply connect to the server, and receive client lists. Currently it has issues consistently parsing data, which we were unable to resolve in time.The flow is as follows. On receiving a connection, the server both spins off a new thread to handle that connection, and logs the client, and tells all threads to update their clients with the client list. At this point the plan is to react to client events. To allow clients to talk to each other, the server also supports forwarding, and will forward a message from one client to another. This reduces the complexity of the clients and lets the server handle all the real work. This is a flexible approach that due to its non-blocking, multi-threaded, design, can react quickly to new events.

Design

The design is simple. The server simply listens on the given ip at a set port (currently 8080) for new connection. On receiving a connection it creates a new *Server_Connection* class, which subclasses Thread so it can run in parallel. The main thread also adds the client information to a list of clients, and sets an atomic flag that all threads check that there is an update in the client list. If an existing thread sees this flag has changed, it updates its client and marks that it did so. The last expected thread (existing at the time of the update flag setting) resets the flag itself. To prevent waste, if a thread finishes it sets an internal flag that the server checks in its main loop, and if it is set the thread is joined and the class is destroyed to save on memory usage. The clients each keep a list of other clients that they update when they receive

a new update from the server. As there are no concrete plans yet, the clients try to test the forwarding feature of the server to talk to each other. This means that they can send a message with a flag set that indicates that the message should be forwarded to the encoded address. The receiving thread then uses the server to pass this to the correct thread so it can forward it to the appropriate client. This fulfills all expected design goals.

Challenges & Open Issues

Currently the server/client fails to properly handle missed parsing of messages, and multiple messages in the buffer may contribute to corruption. Due to a lack of robust watchdogs and reasons to talk to the client,, threads may not realize they are dead for an extended period of time. In the same vein clients may fail to terminate. Right now there is a weak watchdog test to prevent infinite running. Another issue is for some reason the server reports that forwarding succeeded, but the other thread never finds the message to send it. The reason for this is unknown. Currently this program shows promise, and does compile and run, but is not in a working state.