CS2510 A1

Michael Boby

1 Design

1.1 Server

The chat server is set up to run on multiple threads of different types: a single main thread, a single sender thread, and multiple receiver threads.

The main thread of the program sets up the network. It then spawns the main sender thread before entering an infinite loop where it accepts client connections, spawning off receive threads for each connection accepted.

The receiver threads are unique per socket connection (one client may have multiple connections and thus multiple send threads). The receiver thread is passed the client's connection when it is created. Its first task is to initiate a "handshake" with the client. Handshaking consists of receiving a connection request message and then sending back a response stating that the connection was either accepted or refused due to invalid client id. If the handshake is refused, the server terminates the connection. Otherwise, it "fast forwards" the client by sending them all message that would have been received since they were last connected. Then, the thread enters a loop where it receives messages from the client and inserts it into a thread safe job queue for the sender to pull from.

The sender thread is a central thread that broadcasts messages to other clients. It loops, pulling a message off of the job queue and loops through each connection, sending it to each connection whose client id does not match that of the sender (so that the sender does not receive their own message).

1.2 Client

The client program runs on two threads–a sender thread and a receiver thread. The sender thread is responsible for receiving input from the user and sending it out over the network to the server. The receiver thread receives message from the server and prints them out.

2 Implementation

Both the client and the server are written in C using standard sockets for networking and pthreads for multithreading. The only (non-system) external library used was the UTHash library for its hash table implementation.

3 Evaluation

Four tests were written in Python in order to show correct behavior under various circumstances.

Concurrent The concurrent test has three clients. Client 1 listens while clients 2 and 3 send multiple messages simultaneously. Interleaved output is expected, though the order is non-deterministic due to various factors such as network latency and thread scheduling.

Fast Forward This test shows that a client connecting late will receive messages that it has missed. Client 1 connects and sends messages before exiting. Then client 2 connects, receives those messages and sends its own messages before disconnecting. Finally, client 3 connects and receives all messages from clients 1 and 2.

Invalid Login This test shows that a client with an invalid id is rejected. Its expected output is client 1 connecting and sending message before disconnecting, client 2 connecting and receiving messages from client 1 and sending messages, and client 3 attempting to connect, getting rejected and not receiving any messages.

Reconnect An extension of fast forward, reconnect shows client 1 connecting, sending messages and disconnecting, client 2 connecting and receiving those messages, sending some of its own and disconnecting, and then client 1 reconnecting and receiving messages from client 2.

4 Discussion

There are two main issues with the implementation. The first is that the server does not reap old pthread_t tags. The linked list that stores them is only ever appended to and stale entries are never removed. This would not be entirely trivial to correct as there would need to be a way to signal a thread to reap entries. This could likely be done by adding a thread with a job queue similar to the message queue which receives tags to reap.

The second major issue is that all network calls assume fatal errors on a return value of -1. That is, `errno` is not checked to see if the error is recoverable or not. There are a nontrivial amount of network calls in both applications and addressing this would be possible but time consuming.