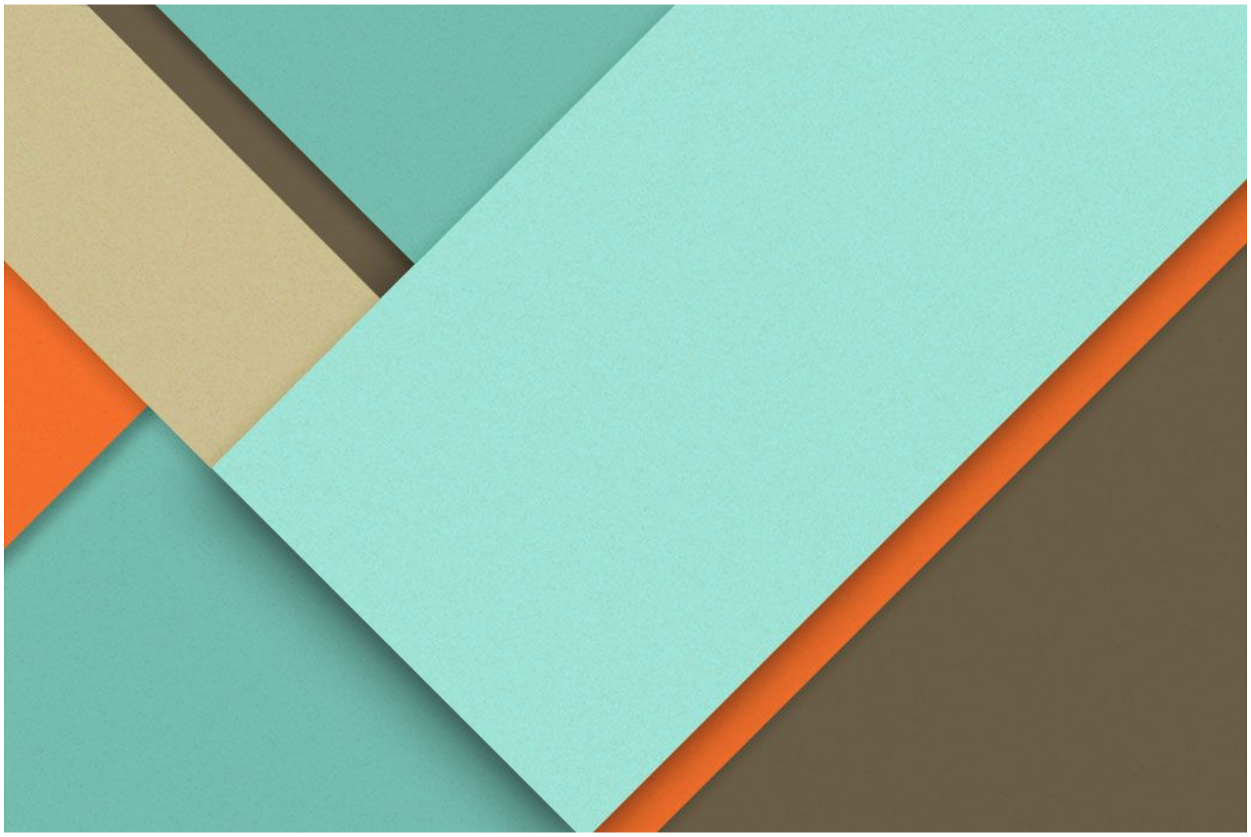


Chatterbox

05.05.2017

Benjamin Shields
CS 422 - Parallel and Distributed Programming
Final Distributed Project - demo report

Overview

Chatterbox is a Java-implemented application for quick chatting in a shell environment. Sockets enable users to join a peer-to-peer network after making a quick request with a central helper server called the Locator. After that point, peers are responsible for growing the network, sending network signals, and passing messages in through a loop of connections to each other.

Goals

1. Use message passing to support basic chat functionality between multiple machines.
2. Have each machine use multiple threads to provide responsive behavior and hide the delays of network communication from users

Logging In

Each new process first connects as a client to the Locator process. The Locator then sends back which port number in the chat network the new user can connect to in order to join. Alternatively, if the new user is the first user process, the Locator tells them this, the new user starts up the network, tells the Locator which port they are 'Listening' on, and then waits for other users to join.

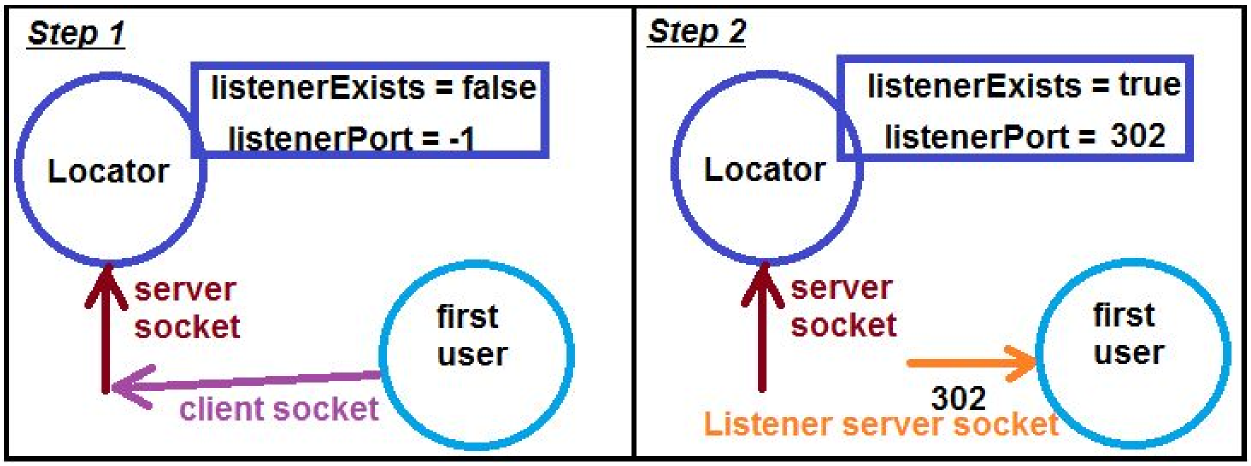


Figure 1, the steps in the first user logging in and starting the network

Listening

The chat network always has one process that is acting as the 'Listener'. The current Listener has a thread running with a server socket, waiting to bring the next new user into the network. After the Listener's server socket accepts a connection, it sends the new user the list of online usernames. Then it waits to receive the new user's choice of a unique username. Next, the Listener aides the new user in joining the chat connection loop as a node in front of itself. This is accomplished by a series of communications to have the new user connect its forward connection to whoever the Listener's forward connection was. Then the Listener's forward connection connects with the new user's backward connection. Finally, the new user communicates with the Locator to assume the role of the new Listener.

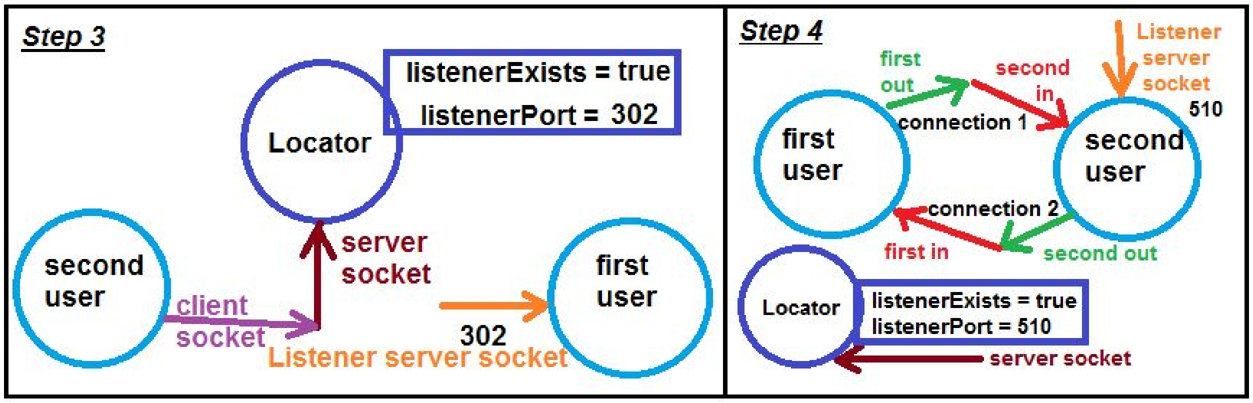


Figure 2, the steps in the first user growing the network after a second user joins

After two users have joined, text messages can be sent and received between them. Step 4 shown in figure 2 shows red and green arrows. When *n* users are online in the network, only *n* connections exist, but each user has two object streams, one input and one output. These are what the green and red arrows represent, two streams per connection.

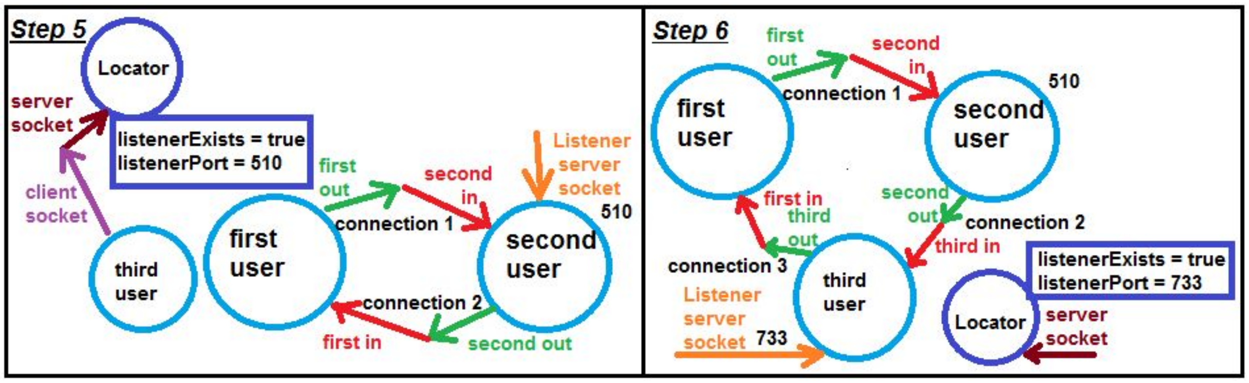


Figure 3, the steps for adding the third user also show how the n'th user joins

Chatting

Each user has a thread running for handling user input from standard in. Once the user is established within the chat network loop, they can begin chatting. This consists of a prompt for the username they would like to send a message to, and then a prompt for the body of the message. Messages are passed through the network by being sent through each node's forward connection, and being received by the next node's backward connection. This communication is unidirectional, forward connections being output-only, backward connections being input-only. These connections are ObjectOutputStreams and ObjectInputStreams. When a node receives an object, it checks if it is a message addressed to them, and if so, displays the message's contents to standard out. Otherwise, the message is passed forward.

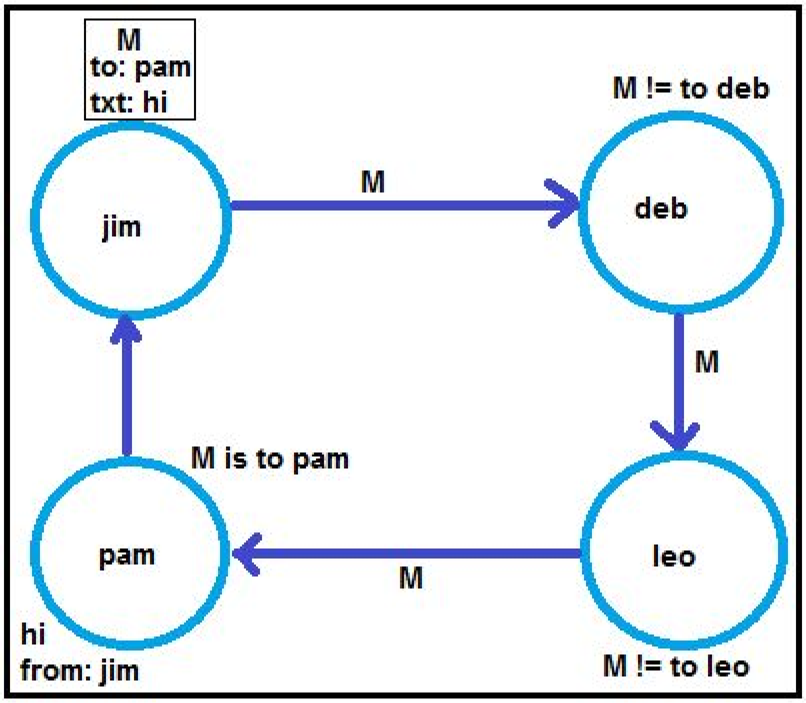


Figure 4, simple message passing in a loop

Usage

The README.txt only applies to running the d2l code on one machine. For new code:

make all (on any shell)
java ListenerLocator (on 930 lab 'harness' machine)

java Chatterbox (each instance starts a new user, tested on 930 lab machines)

Synchronization

Text messages and network update commands are sent using synchronous message passing, blocking until messages are received by the next node (but not necessarily the ultimate intended receiver). But these delays are somewhat hidden in that nodes and users can always write new messages and issue new network update commands (grow and join loop, login, etc.) because each process sues multiple threads. One thread accepts user-input and passes it along to the user's Network object. The Network object then starts up a thread just to send that message along. Network objects always have a Receiver thread running to read objects coming in through their ObjectInputStream. And the user's Network object that is acting as the Listener always has a Listener thread running to accept connections from the next new user.

What I have learned

1. When given a short deadline, I need to focus on the primary task, adding in multithreading and spending so much time designing the interface and encryption ate into valuable time
2. Combining multithreading with distributed processes is incredibly dynamic, interesting, and fun! With the concepts and techniques I have learned in this class, I have vastly increased the domain of problems I can build solutions to
3. It is difficult to pass the listener role around without a central server, but I think that if I use IP addresses as a command-line argument, and always alternate the listener between 2 different ports, I can entirely eliminate the need for the locator

Future Extensions

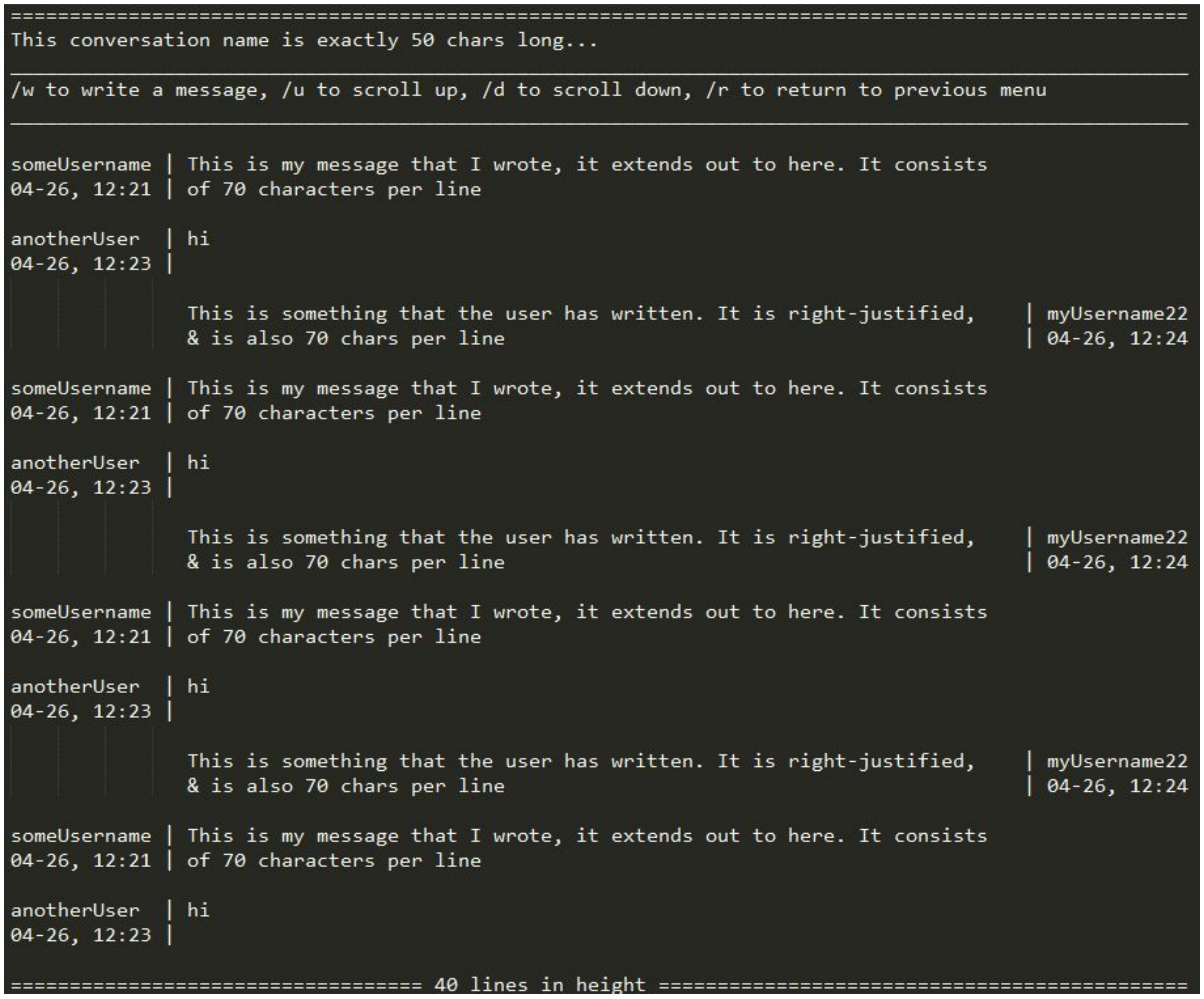
As I was overwhelmed during the time we had to work on this project, I did not finish nearly as much as I hoped to. So, the following limitations exist:

1. All processes must be run on the same machine, I did not yet extend the Locator to also pass the IP address of the Listener
2. Users cannot log out, the required methods were nearly, but not fully implemented nor tested in time

Over the summer I will extend this project to address the above limitations, as well as:

a) Implement an interface that allows users to join one-on-one conversations and group-conversations

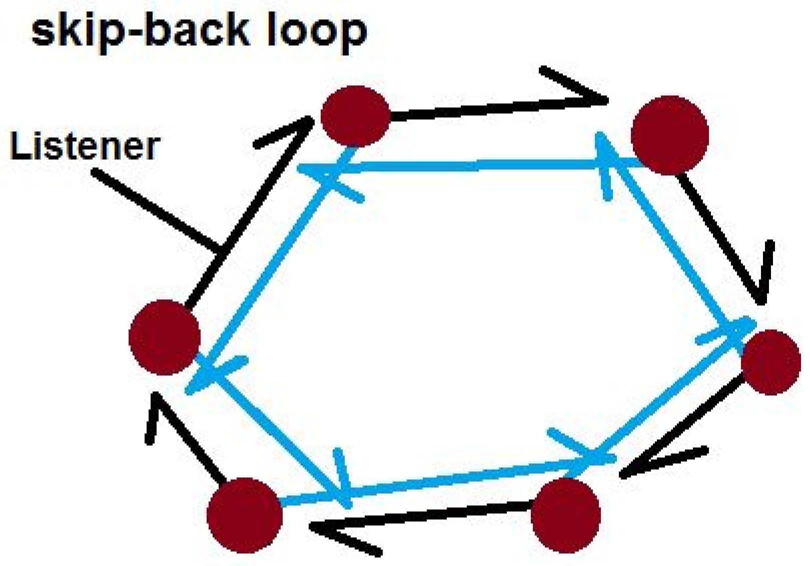
b) Format the appearance of conversations to adhere as follows:



```
====================================================================================================
This conversation name is exactly 50 chars long...
____________________________________________________________________________________________________
/w to write a message, /u to scroll up, /d to scroll down, /r to return to previous menu
____________________________________________________________________________________________________

someUsername | This is my message that I wrote, it extends out to here. It consists
04-26, 12:21 | of 70 characters per line

anotherUser  | hi
04-26, 12:23 |

               This is something that the user has written. It is right-justified,    | myUsername22
               & is also 70 chars per line                                            | 04-26, 12:24

someUsername | This is my message that I wrote, it extends out to here. It consists
04-26, 12:21 | of 70 characters per line

anotherUser  | hi
04-26, 12:23 |

               This is something that the user has written. It is right-justified,    | myUsername22
               & is also 70 chars per line                                            | 04-26, 12:24

someUsername | This is my message that I wrote, it extends out to here. It consists
04-26, 12:21 | of 70 characters per line

anotherUser  | hi
04-26, 12:23 |

               This is something that the user has written. It is right-justified,    | myUsername22
               & is also 70 chars per line                                            | 04-26, 12:24

someUsername | This is my message that I wrote, it extends out to here. It consists
04-26, 12:21 | of 70 characters per line

anotherUser  | hi
04-26, 12:23 |

=================================== 40 lines in height =============================================
```

c) Give users private and public keys to each conversation they participate in, allowing for end-to-end encryption of messages. Intermediate nodes will check if they are an intended receiver by checking hashes rather than usernames

d) Use the following scheme for network connections of peers, giving each node two input and output connections/streams



e) The above connection scheme allows greater flexibility in sending and receiving messages by providing many more paths a message can take. Graph traversal will allow messages to take a randomly-generated path through nodes (without repetition) to obfuscate who the original sender and the intended receivers are

f) Breaking messages into chunks, partly because it increases their encryption and path obfuscation, and mainly because it seems like an interesting problem