CS 262 Design Problem 1
Lab Notes

Initial Thoughts

Client side

→ order of operations when logging into terminal

→ login or create account?

login

→ check if user exists

→ if not

→ otherwise login

create account

→ check that username is not taken

→ login

Display intro text:

→ /help command

→ /display users command

→ / connect [username]

Once in chatroom, can freely type messages

If unread messages:

"you have unread messages from [username]
type /connect [username] to see chat"

Serverside

Keep global dictionary

connections :

[clients queue {'c1': {'c2': [..., ...]}} }]

(loggedinclients {'c1', 'c2', 'c3'}]

len

A broadcast function

→ send messages according to clients and loggedinclient global dicts

Handle function

→ processes requests for individual clients on seperate threads

1) CONNECT USER

→ update connection {'c1':, 'c2':}

→ listen for messages

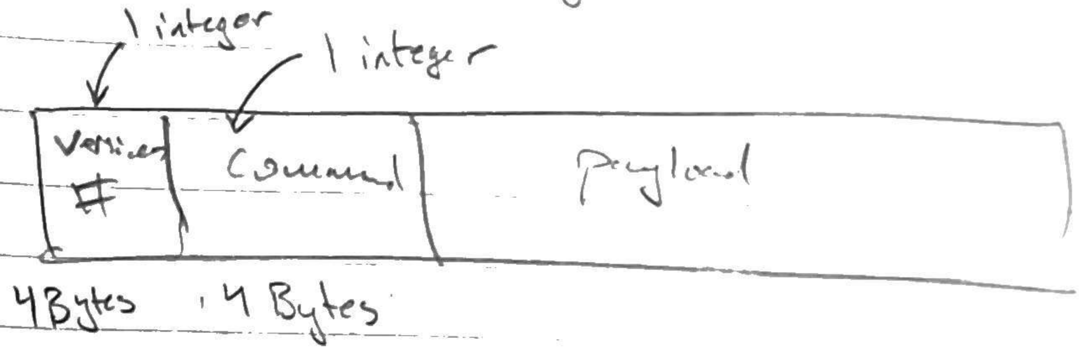
→ check for messages, display them, delete them.

→ send message (client, message)

↳ connection [client] & send message.

add it to

Wire Protocol to allow commands to be sent/recieved along with data



'1' → create
'2' → login
'3' → ...

2/10 : - Cleaned up client interface
- added /show users, /help, and /delete account functions
- have not specified exactly what to do if deleting accont with Undelivered messages, but will come back to this after implementing /connect user and chat functions

Next steps:

Allow two connected users to chat.

→ /connect [username] alters the serverside "Connections" global dictionary, and then allows messages to be queued. Have to think about exitting a chat room

Chatroom logic

→ user logs in and connects to someone
→ start write thread
to simultaneously write and
receive messages

2/11 Update:

- Split work into getting connected
across machines + manual
and automatic testing
→ next steps
- make UI better
- incorporate gRPC

Testing

2/17

→ Connected two separate machines,
Server computer must turn
firewall off, and server's
ip must be known by clients

2/17

gRPC

- ran tutorial
-

TO FIX

- notify receiver about new messages
- check
  - can't send to deleted acounts
  - formatting things (\n)
- we were displaying command # somewhere
- sending super long message (ask for shorter messages) >1024 bits
- delete account while connected to someone

UNIT TESTS:

1) ACCOUNT CREATION
   → existing username (unique?)

to write:

- README file
- engineering notebook

→ have to close write threads

→ text_grpc, check if users are Connected and send respective texts.

If user is not connected but logged on, send notification?

If user is not connected, add to queue

USERNAMES → list of active usernames

LOGGED_IN → usernames that are logged in

Clients → dict to map client to (username

Connections → dict to map connected users in chatroom

queue → message queue if users not connected.

What happens if account is deleted mid Connection?

2/18:

Remaining todo:

- Implement notification of messages from another chat, both normal & gRPC
- What happens when one user closes session mid-connection?
- What happens to queued messages, connections, clients, usernames, when account is deleted?
- UNIT TESTS!!

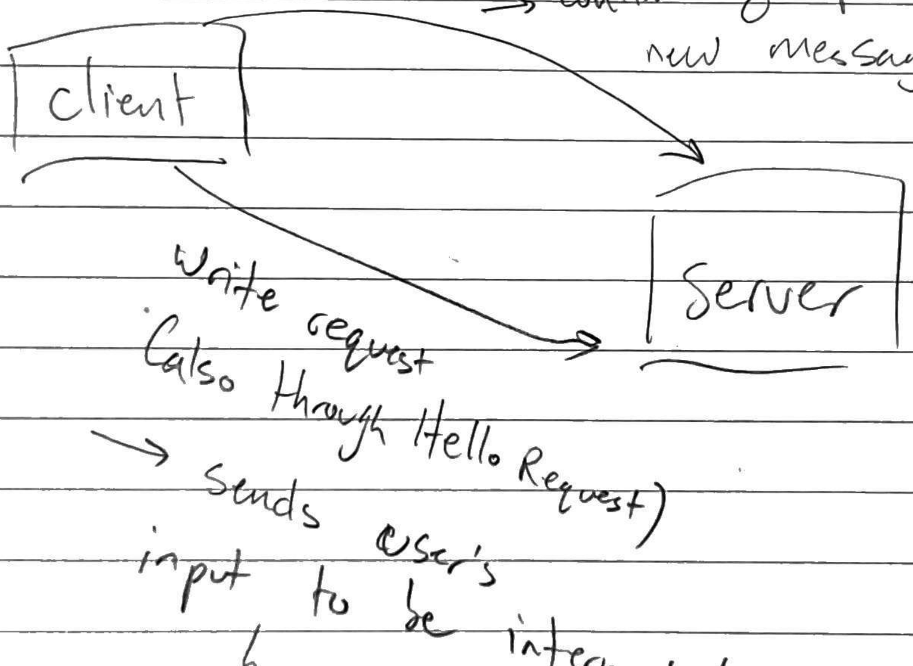
  → how to use unittest package

  → create list of test for both normal and gRPC
- get gRPC version to connect across machines

gRPC design:

client creates unique ID from current time + random number.
Login process is the same, except client always initiates request for response

default HelloRequest function from gRPC documentation examples is the only type of request defined, and we continue to use a similar metadata protocol to the socket version



read_request (through HelloRequest)
→ continuously polls for new messages

Write request (also through HelloRequest)
→ Sends user's input to be interpreted by server

gRPC vs. Sockets Comparison

Code complexity:

Due to the small scale and low functionality of our simple messaging service, We actually don't see a large difference in code complexity. Many of Python functions used to process client commands with pure sockets are similar to ones we use for gRPC - the main difference lies in only the communication mechanism. The wire protocol used for sockets is, instead of converted to bytes, just sent as a string through the gRPC request response object.

Performance differences:

- client continuously requesting?.

size of buffers?.

2/21 updates

for socket

→ both socket & gRPC, make so two clients cannot log in with same username

→ fixed some printing errors

→ Send notification to client if there are unread notifications from a specific person

TODO → fix weird printing for gRPC

→ more tests (multiple users, deleting accounts mid connection)

→ clients cannot connect with same username

→ peformance tests?

→ measure buffer sizes

→ time elapsed?

→ can't do large scale because limited by number of threads

2/21

① → exception for when messages are too large

② text wildcard
- implement some search * in /show
- not allow * in usernames

③ more unit tests

④ GRPC ~~check over multiple computers~~
↳ unittests for GRPC

⑤ update engineering notebook

⑥ performance testing → size of transfer buffer