For the first design exercise, you will develop a simple chat application. This will be a client/server application, with the following functions:

1. Create an account. You must supply a unique username.
2. List accounts (or a subset of the accounts, by text wildcard)
3. Send a message to a recipient. If the recipient is logged in, deliver immediately; otherwise queue the message and deliver on demand. If the message is sent to someone who isn't a user, return an error message
4. Deliver undelivered messages to a particular user
5. Delete an account. You will need to specify the semantics of what happens if you attempt to delete an account that contains undelivered message.

- Allow multiple clients to connect to server simultaneously
- Single server needed at any point
- Code.harvard.edu (or .tar/.zip)
- README
- Documentation
- No names

Lower level: socket dance and connection acquisition is first step. Then, the wire protocol kicks in as the data that flows in/out through that connection.

Client Lifecycle

1. (optional) create account
2. Login
3. Send/receive messages
4. Delete account

Wire Protocol

Account Creation Workflow

Client: CREATE_ACCOUNT

(This is a request to create an account. If you get a confirmation from the server, you're good. Otherwise not.)

- Unique username
    - response/error message if not unique [accept/decline username]

Server: CREATE_SUCCESS/CREATE_FAILURE

- A message sent from the server confirming/denying account creation of a specific username

Login workflow

Client: LOGIN_REQUEST

- Username
- (password in real life but not in the pset)

Server: LOGIN_SUCCESS/LOGIN_FAILURE

- Login success/failure

List accounts workflow

Client: LIST_ACCOUNTS

- Contains an optional text wildcard

Server: ACCOUNT_LIST

- Returns a list of accounts that match the wildcard (if supplied, else all accounts)

Messaging Sending Workflow

Client: SEND_MESSAGE

- Recipient username
- Message data

Server: MESSAGE_FAILURE/MESSAGE_PENDING

- Failure or pending only. Success comes when recipient confirms.
- Pending if recipient isn't logged in, failure if doesn't exist.

Message receiving workflow

Server: DISTRIBUTE_MESSAGE

- Username of recipient
- Data (message itself)

Client: MESSAGE_RECEIPT

- Confirmation that a specific message was received
    - Maybe messages need a unique ID so this goes smoothly
    - Some hash?
- Triggers a MESSAGE_SUCCESS msg from server to original sender.

Account deletion workflow

Client: DELETION_REQUEST

- Username to be deleted (is this redundant w/ sockets?)
- Force flag

Server: DELETION_SUCCESS/DELETION_FAILURE

- Error if requestor is not the owner of account to be deleted
- Error if account to delete does not exist (maybe the same thing in the abstract)
- Error if requestor did not use force flag and there are undelivered messages