Nathan Ackerman
Assignment 1 Report

Usage:
SERVER
python3 server.py -client_ids 1,2,3 -port XXXX

CLIENT
python3 client.py -client_id 1 -server_ip <ipaddress> -port <port>
input for clients can be from a file, redirected through stdin
msgs can be sent from command line after client is started by typing msg and hitting enter

for automated tests:
./run_test1.sh <port_num> <server>
./run_test2.sh <port_num> <server>

DOCKER
from the directory with all of the files:
build with "docker build -t dockerpython ."
run with “docker run dockerpython”
This will run both of the tests inside of the container

Design:

The basic design of the system revolves around a client logging in and pulling updates, and new updates being pushed from the server if the client remains online. Upon login, the server will give the client any messages they haven’t seen by keeping track of the last message id seen by each user. This user on login is mapped to the User object in the server, or one is created if they are logging in for the first time. The server keeps track of any Users that remain online, taking tem offline if the socket connection is closed. If a message is sent into the system, any online users are pushed that message and the message is stored to update users that log in later.

The server process is an infinite loop that accepts connections. A new thread is spawned to deal with each user connection. Upon verification of the user against the valid client ids, the thread will update that user and then listen on an infinite loop for messages from that users socket. Upon receipt of a message, the thread will send to any online clients (not the sender).

The client process has two threads. After verifying login, one thread is spawned to listen for updates from the server of new messages in an infinite loop. The other thread is for the user to type messages and send to the server by hitting enter.

In general, communications from the server and client happen via 1024 byte buffers. These buffers may contain multiple messages to reduce on communication costs. Being a multi-threaded server, many of the variables are guarded by locks to ensure correctness.

Possible Improvements:

One improvement would be using a database to store the messages at the server. This would increase the fault tolerance and allow the server to operate on the same data across multiple runs/crashes. As it stands, the messages are kept in main memory, meaning if the process dies, memory corrupts, or machine goes down, all messages are lost. Stable storage would prevent this.

Another improvement would be to not use specified ports for the server. I did it this way because the example in the assignment had it being done this way, but this is very bad practice. There

are better mechanisms that ask the OS to assign a port on bind. This avoids the situation where ports are already in use by another process on the system.