Chris Gross - cjg28
EECS 325 - Computer Networks
December 9, 2011

Project 5 - Messaging Client

Application Description

The purpose of this application is to provide the user with a way to message another user running the same client. The way this works is by setting up a server which contains all the information on users. By doing this the client does not need to know any information about the person he/she wants to talk to aside from their name. The client also runs a server which is forked off as another process from the main process which talks to the IM server. This allows for the user to receive messages indefinitely. This server is also known by the IM server as it logs both IP and port information for the user.

Usage Instructions

Server

The server must be run to allow for communication between clients.

1. To run the server type **./proj5 <port num>**
2. The server will run indefinitely so you must kill the process with **Ctrl-C**

Client

The client assumes that all necessary ports are open and tries to find an available port starting at the range defined in **constants.h**. If it can't find a port to leverage it will not run. ensure port forwarding is enabled on your router.

1. To run the client type **./proj5d <hostname> <port num>**
2. Once the server is run you must type in a valid user name.
3. Then the command line will launch allowing you to type commands.
   Commands
   i. **message**: Messages a user
   ii. **port:** Changes **your** chat clients messaging port
   iii. **username**: Changes your established username
   iv. **disconnect:** Disconnects and ends your session
   v. **chat:** Allows you to stop entering commands and just type messages to a user until the command **<end chat>** is typed in.
   vi. **list users:** Returns a list of all currently logged users.
   vii. **<end chat>:** Ends the chat session and returns to command mode.
   viii. **help:** Displays all these commands and their definitions.

Application Protocol

Client-Server

The way this application communicates between the client and server is through the usage of a command byte located in the messages sent. This command byte is located at byte **2** as defined in **constants.h (COMMAND_LOCATION)**. There are different types of command bytes for every command sent and these are also defined in **constants.h**. A message must have a command added to it using the function

addCommand(char * buffer, int command)

which adds a command as defined in the **command** field. Additionally, once this command has been read the message can be stripped using the

grabContents(char *buffer)

function. Once the server has read a command it returns a message to the client which reads and parses the message according to any fields required in the command location byte field.

i. **message (3)**: Server returns that username input is valid and the IP and port to reach the user at. then the client forks a process to send the message.
ii. **port(4):** Reads port input and changes youre reference port
iii. **username(-1)**: Changes your established username as based on your sock location in the users array.
iv. **disconnect(1):** Disconnects and ends your session, returns a disconnect call to client
v. **chat(5):** Same as message but doesn't fork.
vi. **list users(2):** Returns a list of all currently logged users.

Note: Server will disconnect user on receiving an empty message from the user (no \n).

Note: The server logs user IP and port on initial connection (even before username is established)

Client-Client

Communication between client and client is done through the forked servers which each client runs. The client messaging server forks off a new process every time a message is received and handles it based on message type (NOT CHAT). If it senses a **chat byte** in the **COMMAND_LOCATION** field the forked process will loop until it receives the end of chat call.

Transportation Protocol

This application uses TCP as opposed to UDP. This is because UDP does not provide reliable data handling which would cause extra overhead on the receiving and sending of messages from all clients and servers. Additionally, this ensures that ALL data will be sent as requested by the application.