Lab 2 report

Felix Sjöqvist and Olle Olofsson

April 28

1 Overview

This program includes a server and one or more clients. The server process is letting one or more clients connect to it through socket programming. When a connection between the server and client is established, they can send data to eachother. More specifically the client process can take user input and send it to the server which will print it out on the screen. Since multiple clients can be connected to the server at the same time, when a new client is connected the server broadcasts a message to all other connected clients, informing them about the new client.

2 How it works

The server creates a socket and assigns it a name and an address. Then the socket gets put in a state where it listens for incomming connections. When a client want to connect the server notice that there is some data/input on its socket accepts the request. A new socket is created, different from the original socket, which then is used for communicating with the client. At this point the server can notice the client's IP-address and decide if it wants keep or reject the connection based on if that specific IP-address is "banned" or not, this is done by comparing the two strings with `strcmp()`. If the server chose to keep the connection, the server informs the client by sending a message "I hear you, dude..." to it, else the server closes the socket and stops the connection. The server has two sets of file descriptors/sockets, a read-set and a active-set. The active-set contains the file descriptors of all active sockets i.e. the server socket and the client sockets produced when a client is accepted. Each time a client successfully connects, the sever is broadcasting out to all the other clients that a new client has connected. This is done by looping through all sockets in the active-set (exept the sever socket) and writing a message to that socket.

The client creates a socket and initializes its address depending on the hostname-agrument the user put in and the port number. It then tries to connect to the the address given which in this case is the server. If the server accepts its connection, it creates a thread which is continously listening for input on its socket i.e. for incomming messages. The main thread waits for input from the user which it will then send on the socket connected to the server. If the user inputs "quit" the client closes the socket and terminates.

3 Program outputs

- Server's output when two clients connect

```
[waiting for connections...]
Server: Connect from client 127.0.0.1, port 57220
>Incoming message: hej

Server: Connect from client 127.0.0.1, port 57222
>Incoming message: hej2
```

- The first client's output when first connecting to the server and a second client connects

```
Type 'quit' to nuke this program.
>Server: I hear you, dude...
>Server: Client 127.0.0.1 connected
```

- The second client's output when connecting to the server

```
Type something and press [RETURN] to send it to the server.
Type 'quit' to nuke this program.
>Server: I hear you, dude...
```

- Output of a client whos name/IP-address is banned in the server.
  (The program lost connection and terminated after the last line)

```
Type something and press [RETURN] to send it to the server.
Type 'quit' to nuke this program.
>Server: You are banned
```

- The server's output from when the banned client tries to connect.

```
[waiting for connections...]
Server: client 127.0.0.1 rejected
```

4 code

- be well commented on important functions and lines of code (undocumented code will not be approved)
- include meta-data on each file, such as program name, author, general description
- answer the question why and now how the program works (the code itself often explains how something is done)