2nd Group Homework Report

TCP Simple Broadcast Chat Server and Client

ECEN 602 Network Programming Assignment 2

Mainly, we worked together, but here is the role what we did.

Minhwan Oh : Developed server programming, debugged for integration

Sanghyeon Lee : Developed client programming, tested for integration

File info

===============

For this program, you can see how TCP Simple Broadcast Chat Server and Client works. There are two main files.

1. Client.c

Path: cd Assignment\_2/Client.c

Main feature: Client application for chat room

2. Server.c

Path: cd Assignment\_2/Server.c

Main feature: Server application for chat room

3. unp.h

Path: cd Assignment \_2/unp.h

Main feature: Packages for socket programming function \*Refers to Unix Network Programming library\*

4. config.h

Path: cd Assignment\_2/config.h

Main feature: Autoheader

5. makefile

Main feature: Compile setting description

Build info

===============

Command $make

Program scenario

===============

This program includes client and server for a TCP simple broadcast chat service, which does the following:

1. Start the server first with the commanding lien: $server <IPAdr> <Port> <Accessible Client Number>, where “server” is the name of the server program, IPAdr is the IPv4 (or IPv6, need to switch several lines in code) address of the server in dotted decimal notation, Port is the port number on which the server is listening, and Accessible Client Number is the maximum number of clients can join server at the same time. The server must support multiple simultaneous connections.

2. Start the client second with a command line: $client <User Name> <IPAdr> <Port>, where client is the name of the client program, User Name is the client’s name which will be used in server as a client ID, IPAdr is the IPv4 (or IPv6, need to switch several lines in code) address of the server in dotted decimal notation, and Port is the port number on which the server is listening. Then, client sends JOIN type to server to join the chat session.

3. When server receives the JOIN type header from a client, the server checks the client’s username is on the existing client list. If the same user name is assigned on the list, the server sends NAK type header to reject its connection. If the user name is new, the server sends ACK type header to the client to confirm the JOIN request and broadcasts the arrival of new participant to the other client who are already in the chat session by sending ONLINE header message.

4. A client sends a message (e.g. Hi guys!) to server, and the server receives the message, then broadcast it to the other clients by using FWD message which contains the message and its username. If header or attribute type is wrong, DISCARD it

5. If a client doesn’t send any message for 10 seconds, the server broadcasts it to the other clients that the user is in IDLE state. However, if the client sends any message again, the IDLE state is over, and the server counts the time again to check the IDLE state of each clients in the chat session.

6. A client leave the session unceremoniously. The server broadcasts OFFLINE message to the other existing clients and cleanup the allocated memory of the left client for new clients can use that user name again.

Program details

===============

1. The communication will work with either IPv4 and IPv6 networks.

1. The server handles all clients’ access by using ACK(accepted) and NAK(rejected) messages and broadcasts the clients’ status (ONLINE(new client joined), OFFLIEN(client left), and IDLE(didn’t move for 10 seconds)) and each other’s messages (SEND and FWD) to other clients.

2. The client ask (JOIN) permission to join chat session and gets the client information of chat session (current client list and the number of client).

3. Clients send and receive messages through server’s FWD system.

4. The server detects clients’ idle status, which clients don’t send any message for 10 seconds, and inform it to the other clients. If the idle clients move, the idle status is over and the server initiates the countdown idle timer to zero.

File architecture

===============

1. Server

a) Architecture

- Scenario :

(1) Setup client\_info/user\_check/broadcast/ACK/NAK/bind/listen functions

(2) Then going to infinite loop until the exit

- Main function :

(1) int main(int argc, char \*\*argv): Including main scenario

(2) int check\_newFd(int nClient, int nMax\_Client, char \*username, Client \*clientinfo,int connfd): Check new client

(3) void broadcast\_all(Message broadcast\_Msg,fd\_set check\_fd, int listenfd, int connfd, int latest\_fd): forward clients’ status and messages

b) Feature

- Create socket and bind the well-known port of server

- Check new clients and accept or rejects them

- Forward clients’ status and messages to the other clients

2. Client

a) Architecture

- Scenario :

(1) Setup JOIN/sendMessage/recvMessage/socket/connect function

(2) Then going to infinite loop until the exit

- Main function :

(1) int main(int argc, char \*\*argv): Including main scenario

(2) int joinServer(char \*buf,int sockfd, int size\_buf): Make JOIN in SBCP message format

(3) int RecvMsg(int sockfd): handle received messages depend on its header type

b) Feature

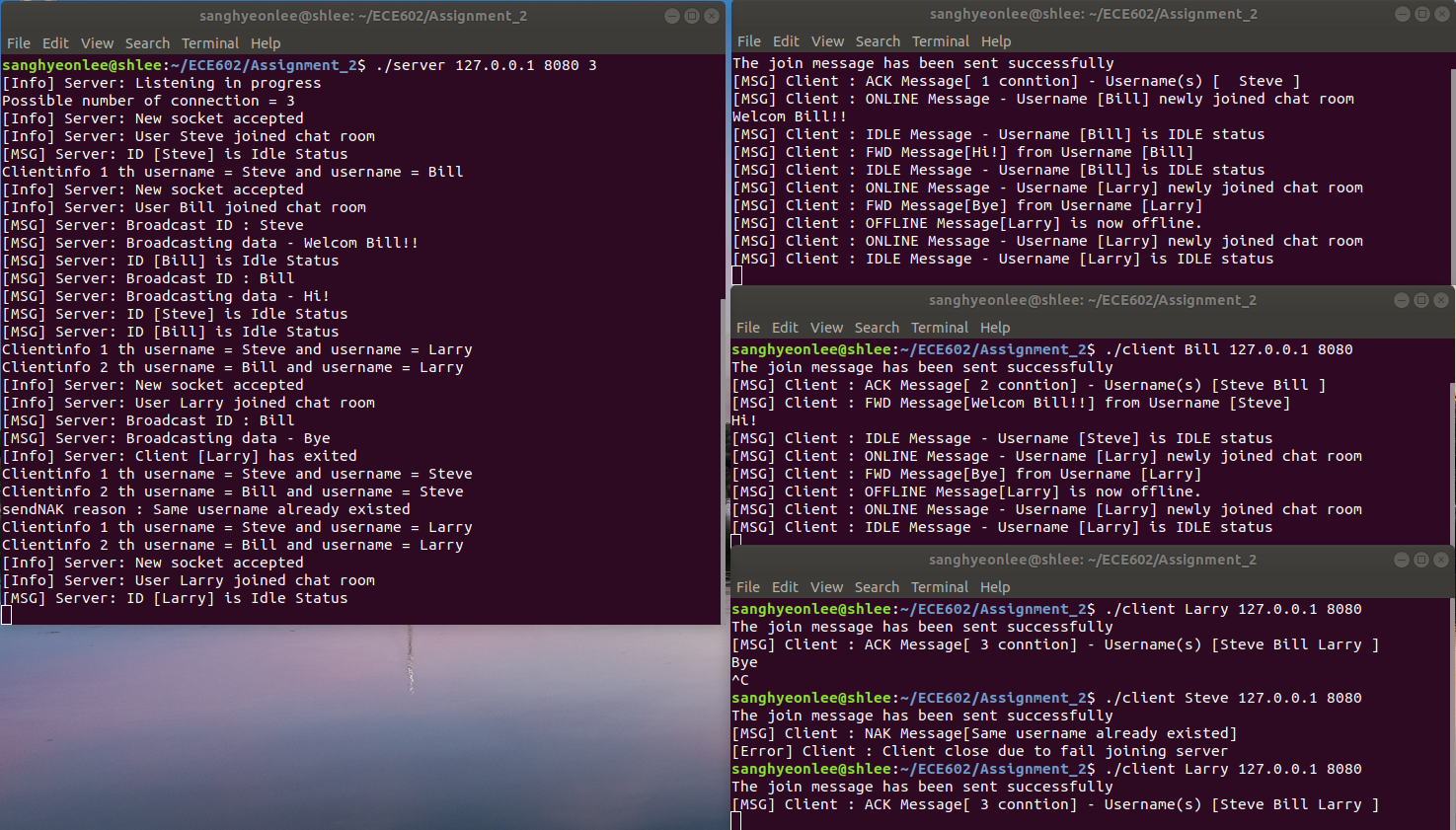
- Create socket and bind the well-known port of server

- Monitor headers in received message and print it.

Result

================

Simulation overview



Requirement

===================

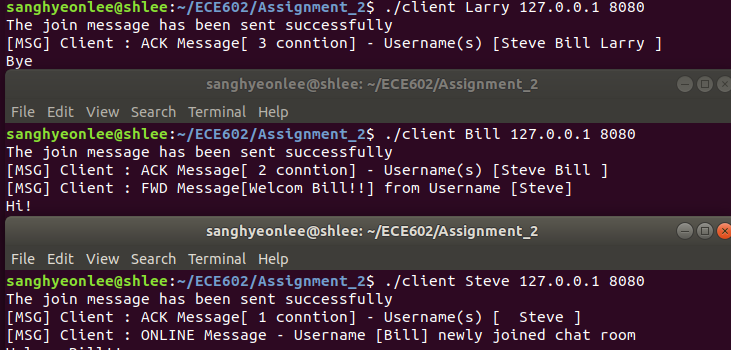
|  |  |
| --- | --- |
| Client | |
|  |  |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Server | |
|  |  |
|  |
|  |
|  |
|  |
|  |

Test cases

===============

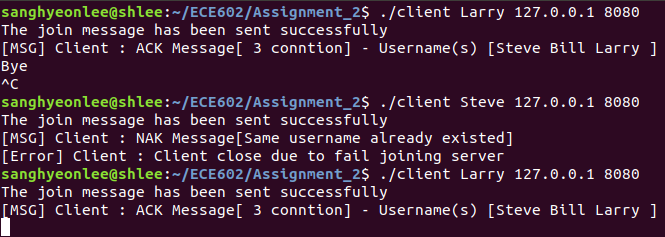
1. Normal operation of the chat client with three clients connected



The maximum number of clients can be set from server input. The clients’ information is stored in client\_info data. The figure above shows that three clients are connected to the server at the same time.

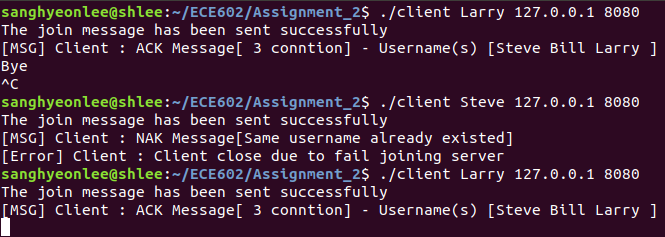
This can be done by I/O multiplexing which makes it possible that send more than two data through one channel. In select function, the server can monitor file descriptors and check buffer whether it’s empty or not by checking FD\_ISSET.

1. Server rejects a client with a duplicate username



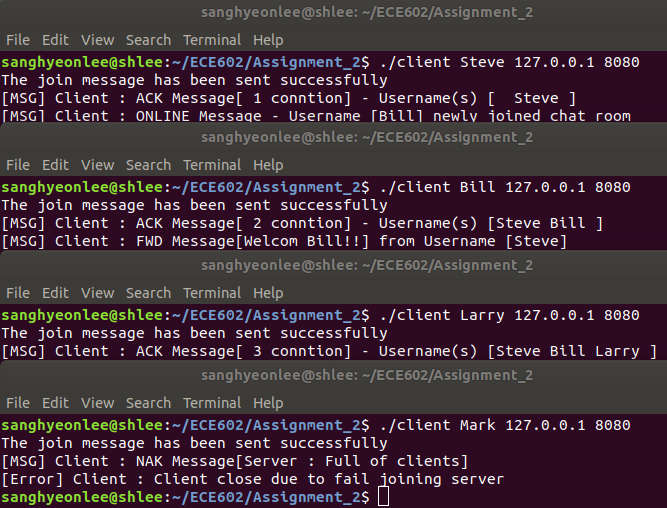
Steve is already on the list of client so that the server sends NAK to the new client and rejected it. Function check\_newFd is where the server checks the new username is already used for current clients’ username.

1. Server allows a previously used username to be reused



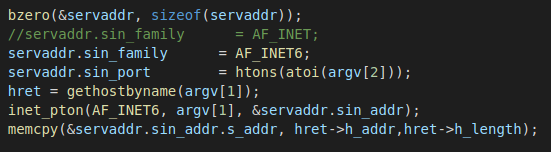
On the other hand, Larry who had left before re-enter the chat session and reused its username again because the username Larry was removed when the client left through a function delete\_clientinfo.

1. Server rejects the client because it exceeds the maximum number of clients allowed.



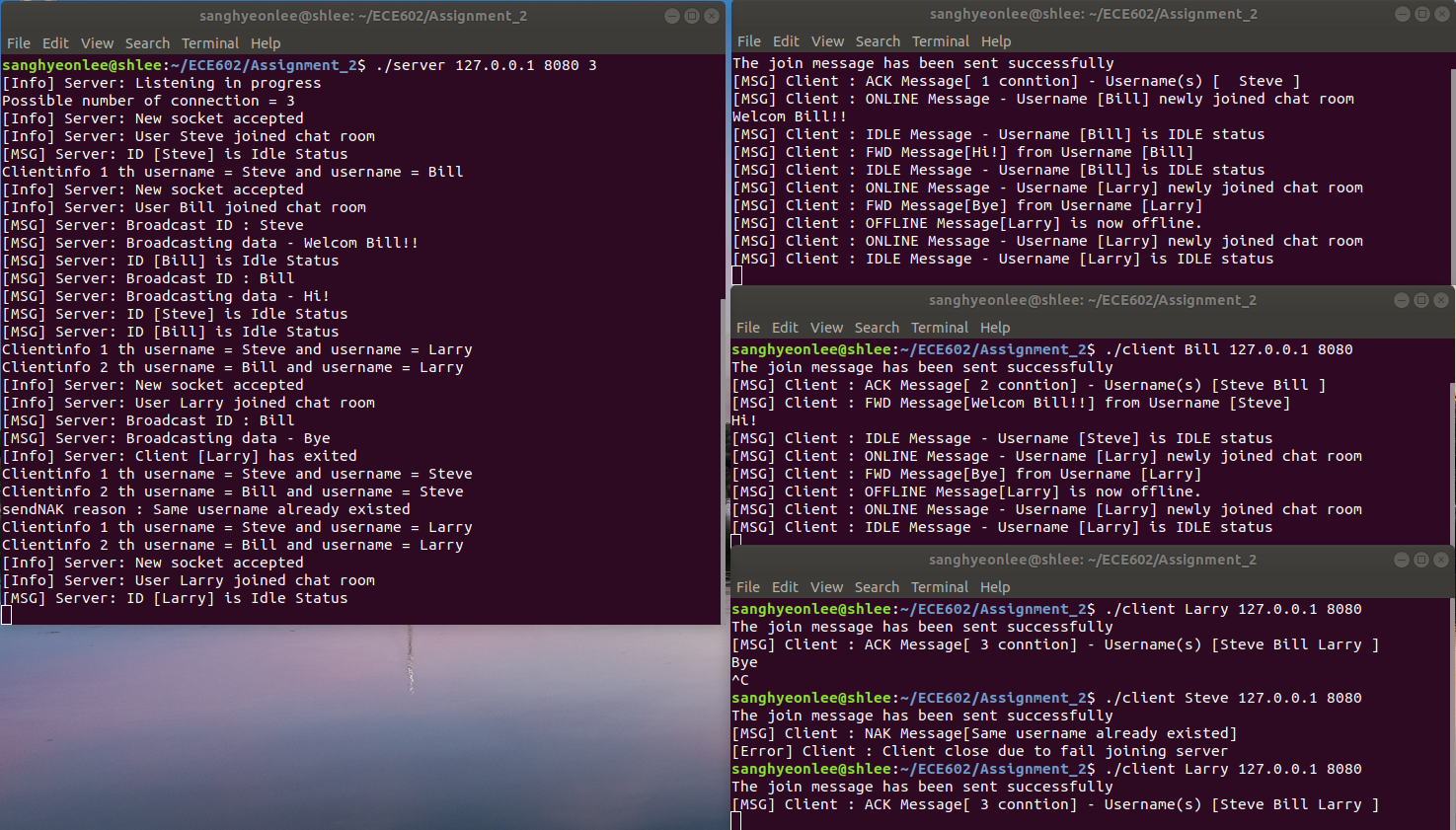
The maximum number of client is 3, and currently three clients, Steve, Bill, and Larry are in the chat session. So, the fourth client Mark has rejected due to “full of clients” reason. This can be done with condition functions with the defined maximum client number.

1. Bonus features
2. IPv4 and IPv6



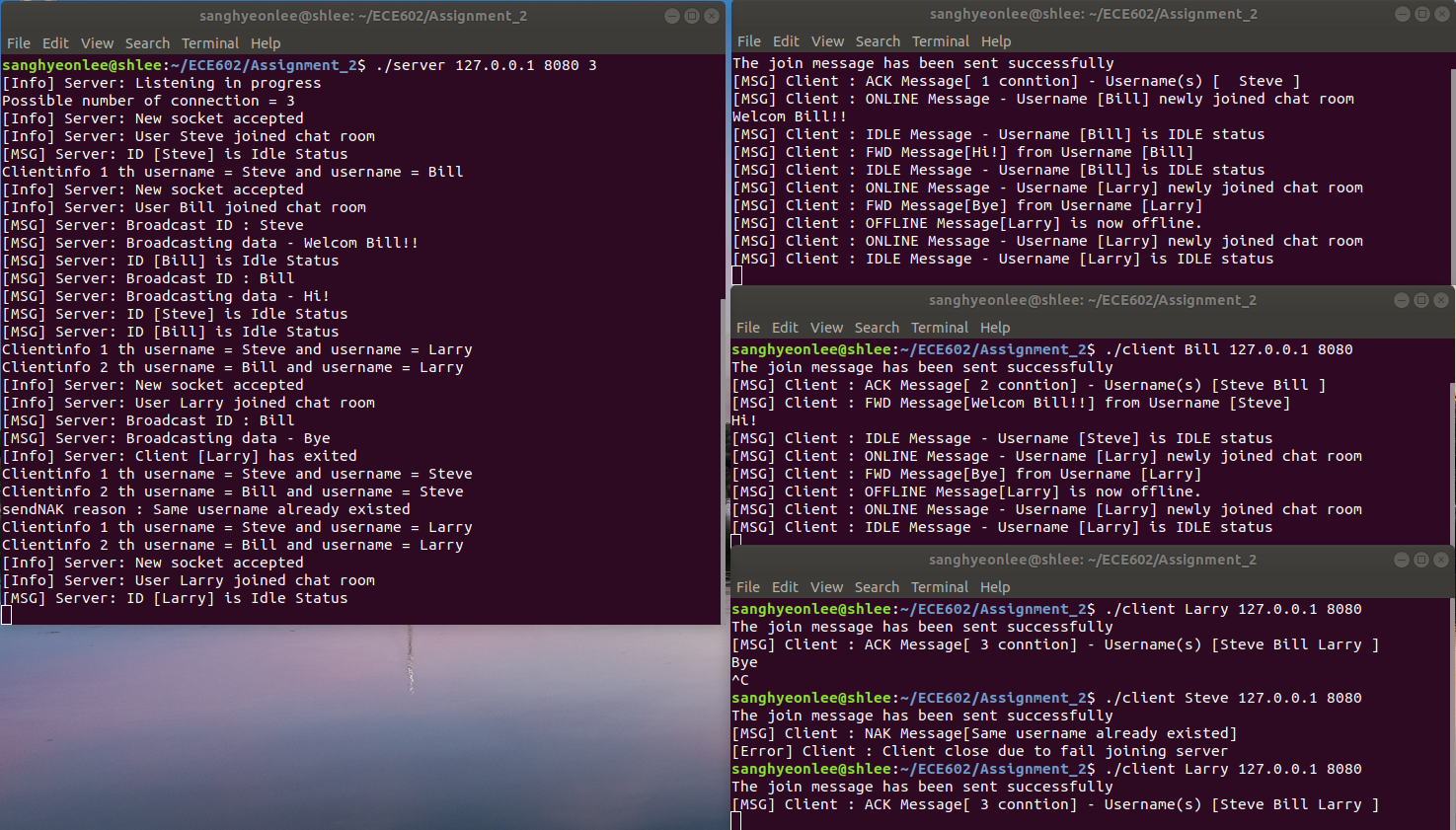
We used Hosten which is compatible for IPv6 and changed AF\_INET to AF\_INET6 and inet\_pton().

1. Feature 1



We made SBCP header types not only JOIN, SEND, and FWD, but also ACK, NAK, ONLINE, OFFLINE, and IDEL. ACK is accept header for a new client to join the chat session. NAK is reject header which returns reasons why the new client cannot join the chat session. ONLINE header is status which is broadcasted to other clients by server. OFFLINE header is also status to inform the other clients that a client has left. Even though client left unceremoniously, server broadcasts the other with OFFLINE header.

1. Feature 2



IDLE is status header. When a client doesn’t send any message for 10 seconds, server counts the time and inform the others that the client is in IDLE status (i.e. sleep mode). Once the IDLE status is informed server doesn’t send it again. However, when the IDLE client send messages, the idle timer initiated to zero. So, if the client doesn’t send any message again for 10 seconds, the server broadcasts IDLE status again.