

Joy Hauser
Eugene Vasserman
CIS 525
31 October 2017

Programming Assignment B

Preliminary Notes

For this project, I decided to use C to implement my solution.

Part 1

Plan

I designed the client to connect to the server directly via file descriptors. The server is keeping track of one file descriptor that is a point that clients can connect to if they want to chat on the server. The server then keeps track of another file descriptor for each new client. I am then using the select command to loop through all the file descriptors and see if anything needs to be done.

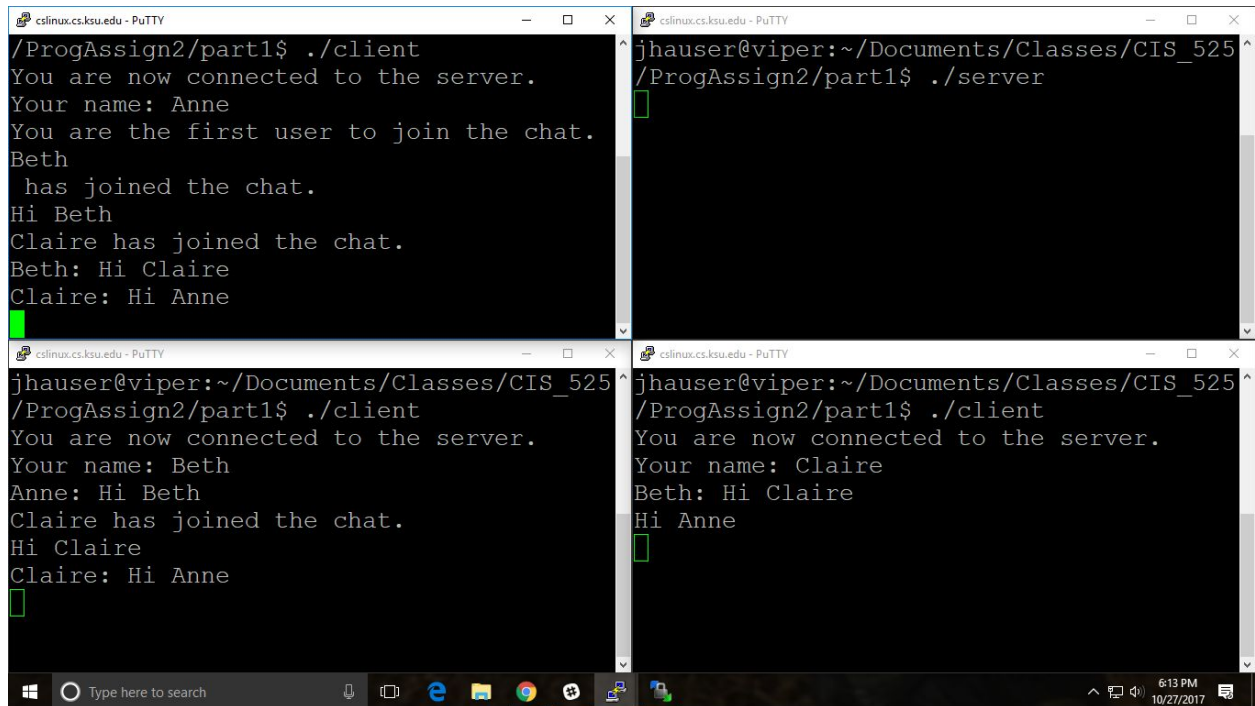
When the select command says that there is a file descriptor that has something to say or do, I determine if that is the file descriptor that is listening for new connections. If so, I am assuming that the person trying to connect is a new client and am adding them to the list of people in the conversation. If it is not the file descriptor that is looking for new connections, there are three types of interactions that happen. The three different types of interactions are:

- 1.) *Specifying a name for the client to the server* To specify a name to the server, I am sending a message from the client to the server that starts with "Name: " and then specifies the name that the user chose. If that name is already being used by another client registered with the server, the server will send back a message that says "bad". Then the client prompts the user to choose a new name. If the name is not currently being used, then the server sends back a "good" message and the program will continue processing as normal. I am also using this field specify if the user is the first user on the server by appending that information to the good message if the name is approved. Below is an example of a normal interaction.
- 2.) *Killing the client* To kill the client, the user needs to press Ctrl+C. When the signal for Ctrl+C, the client will send a message, similar to "Joy: quit", to the server and then kill itself. When the server gets a message that contains "quit", it will break the connection with the client and then continue processing as normal.
- 3.) *Sending a message* To send a normal message, the client will send a message to the server in the format of "<name>: <msg>". If that message does not start with "Name: " or contain the word "quit", it will get forwarded to all the other users on the chat server. I have checks on the client side for both of the special cases.

Results

I completed all the different components of the assignment specified in the prompt document.

Output



The image displays four terminal windows arranged in a 2x2 grid, showing the execution of a chat server and client program. The top-left window shows a client running `./client` and connecting to the server, with messages from the server indicating the user's name (Anne) and other users (Beth, Claire) joining the chat. The top-right window shows the server running `./server` and accepting connections. The bottom-left window shows the client running `./client` and connecting to the server, with messages from the server indicating the user's name (Beth) and other users (Anne, Claire) joining the chat. The bottom-right window shows the client running `./client` and connecting to the server, with messages from the server indicating the user's name (Claire) and other users (Beth, Anne) joining the chat. The Windows taskbar at the bottom shows the time as 6:13 PM on 10/27/2017.

```
/ProgAssign2/part1$ ./client
You are now connected to the server.
Your name: Anne
You are the first user to join the chat.
Beth
  has joined the chat.
Hi Beth
Claire has joined the chat.
Beth: Hi Claire
Claire: Hi Anne

jhauser@viper:~/Documents/Classes/CIS_525
/ProgAssign2/part1$ ./server

jhauser@viper:~/Documents/Classes/CIS_525
/ProgAssign2/part1$ ./client
You are now connected to the server.
Your name: Beth
Anne: Hi Beth
Claire has joined the chat.
Hi Claire
Claire: Hi Anne

jhauser@viper:~/Documents/Classes/CIS_525
/ProgAssign2/part1$ ./client
You are now connected to the server.
Your name: Claire
Beth: Hi Claire
Hi Anne
```

Part 2

Plan

For the chat directory server, I used a very similar process for handling new connections. I have one file descriptor for the connection that anyone can connect to. Then when either a chat server or a client tries to connect to the chat directory server, I am accepting the connection and adding that new connection to list of connections to keep track of. The chat directory server then handles three different types of messages, which I have specified below:

- 1.) *Get directory list* I am keeping a list of the chat servers that are connected to the chat server directory. When I get a request for the directory list, I am putting all that information into a string and passing it to the connection that ask for it. To get the directory list, the connection needs to send the chat directory server a "Get directory" message.
- 2.) *Register chat server* To register a server, the connection needs to send the chat directory server a message of the format "Register server:<server name>;<port number>". The chat directory server will then get the name out of that request and see if that name is already used. If that name is already used, it will send a failure message to the chat server and the chat server will kill itself. If the name is not already used, it will

tell the chat server that it has registered the server and add the chat server's information to list of chat server information.

- 3.) De-register chat server To de-register a chat server, the connection needs to send the chat directory server a message of the format "De-register server". When that message is received, the chat directory server will sever the connection between the chat directory server and the chat server.

On the client side, I made a few modifications to the code I used from part 1. At the beginning of the program, it connects to the chat directory server right away. Then it asks for the directory from the chat directory server and displays it to the user. The user then specifies the server that they want to connect to in the format of "<server name>;<server port>". After the chat server has been selected, the connection to the chat directory server is closed and the program continues as it did for part 1.

For the chat server code, I made a few modifications as well. I modified the program to take in the server name and the server port. Then the chat server will connect to the chat directory server and attempt to register itself. If the chat server receives a success message, it will continue as it does in part 1. If the registration of the chat server fails, the chat server will kill itself. If the chat server is killed using Ctrl+C, it will send a message to the chat directory server to de-register the chat server and then it will kill itself.

Results

I completed all the functionality for part 2 that was specified in the project description.

Output

```
jhauser@viper:~/Documents/Classes/CI5_525/ProgAssign2/part2$ ./client
Start setup of chat server
Before creating socket
Before connection to directory server
After connecting to dir server
You are now connected to the directory server.
Send message to chat dir server
Chat Server Director
test:1234
Select server: test:1234
Server name: test:1234
Port #: 1234
You are now connected to the chat server.
Your name: Bobby
Hello Joy!
How's it going?

jhauser@viper:~/Documents/Classes/CI5_525/ProgAssign2/part2$ ./dir
Someone is trying to connect to dir server
Accept incoming connection
Register chat server
Someone is trying to connect to dir server
Accept incoming connection
Get directory
Someone is trying to connect to dir server
Accept incoming connection
Get directory

jhauser@viper:~/Documents/Classes/CI5_525/ProgAssign2/part2$ ./server test 1234
Client msg: Name: Joy
Client msg: Joy: Hello
Client msg: Name: Bobby
Client msg: Bobby: Hello Joy!
Client msg: Bobby: How's it going?
```