Assignment 07

If any of this is unclear, please get in touch with me ASAP. There are definitely things in the course I want you to struggle with. I absolutely do NOT want you struggling to understand my expectations.

In this assignment, you will implement both server and client for a simple online chat system. Every client will represent a user, which may have a name associated with it. Upon connecting, the name will be "unknown", but a user may change their name by sending the message "`/nick <new nickname>`" to the server. The server will announce the name change to all connected clients. Similarly, any message sent by a client will appear for all clients.

Following are example runs of a single server and two separate clients running concurrently.

Server (all text output by server process, pressed Ctrl-C after first client exited):

```
$ ./chat-server 8000
new connection from 127.0.0.1:46814
new connection from 127.0.0.1:46816
User unknown (127.0.0.1:46814) is now known as foo.
User unknown (127.0.0.1:46816) is now known as bar.
Lost connection from foo.
^C
```

Client 1 (the lines consisting of only "/nick foo", "hi", and "I'm just fine, thanks" were input at the terminal, followed by Ctrl-D):

```
$ ./chat-client localhost 8000
Connected
/nick foo
22:31:37: User unknown (127.0.0.1:46814) is now known as foo.
22:31:41: User unknown (127.0.0.1:46816) is now known as bar.
hi
22:33:40: foo: hi
22:33:42: bar: hello!
22:33:45: bar: how are you?
I'm just fine, thanks
22:33:48: foo: I'm just fine, thanks
Exiting.
```

Client 2 (the lines consisting of only "/nick bar", "hello!", and "how are you?" were input at the terminal, exited when server terminated):

```
$ ./chat-client localhost 8000
Connected
22:31:37: User unknown (127.0.0.1:46814) is now known as foo.
/nick bar
22:31:41: User unknown (127.0.0.1:46816) is now known as bar.
22:33:40: foo: hi
hello!
22:33:42: bar: hello!
how are you?
22:33:45: bar: how are you?
22:33:48: foo: I'm just fine, thanks
22:34:01: User foo (127.0.0.1:46814) has disconnected.
Connection closed by remote host.
```

New Mechanics

You might find these useful in this assignment.

New(ish) functions

- `kill(2)`
- `time(2)`
- `strncpy(3)`
- `snprintf(3)`

(In addition to those functions introduced in the various course notes.)

New machines

You have used `basin.cs.middlebury.edu` already in this course to store your upstream git repos. Unfortunately, as we discovered in class, the firewall on `basin` does not allow arbitrary incoming network connections, and so you won't be able to use it to test your chat server. The good news is that there is another machine available for this purpose: you may ssh to `weathertop.cs.middlebury.edu` and run your servers there. `weathertop` accepts incoming connections on ports 4000 through 4020, so use a port in that range. Recall that multiple sockets can't bind to the same port; you can use the command `netstat -tnl` to see which ports are in use. If you are off-campus, you will need the VPN active to connect to `weathertop`, just as you do for `basin`.

Behavioral requirements

For the server…

- Its single command-line argument should be the port number upon which to listen for incoming connections.
- Every client must be handled by a separate thread.
- When a client connects, the server should print a message to stdout identifying the new client by IP address and port number.
- When a client changes its nickname, the server should print a message to stdout indicating both old and new names.
- When a client disconnects, the server should print a message a message to stdout indicating so.
- Must support an arbitrary number of clients connected simultaneously.

For the client…

- Takes two command-line arguments: the hostname and port number of the server.
- Upon receiving a message from the server, prints it out preceded by a timestamp.
- It is acceptable if the terminal becomes garbled when a message arrives while the user in the midst of typing their own message.
- If the user enters end-of-file (ie, the user types Ctrl-D), the client should terminate with a descriptive message.
- If the connection is terminated (eg, the server process terminates), the client should also terminate, with a descriptive message.

For the system…

- When a client sends a message, every client connected to the server should receive it and print it out, with proper attribution.
- When a client disconnects from the server (for whatever reason), the server should notify all remaining clients, which will print an informative message.

Deliverable requirements

- `chat-server.c` and `chat-client.c`, which contain the code for the server and client, respectively.
- `Makefile`, whose default target should build `chat-server` (out of `chat-server.c`) and `chat-client` (out of `chat-client.c`). It may have as many other targets as you like, to make your life easier), but the default one *must* build those programs.
- `README`, which will include a list of authors, a list of known bugs, and a list of resources you consulted in your work (a list of URLs is totally acceptable—if I can see the kinds of references you're using, I can get a better idea of how to tailor the instruction to your needs—just keep the README file open as you work and paste them there when you find something useful).

Style requirements

Your code must exhibit:

- appropriate use of comments;
- appropriate use of stack, heap, global, and static variables;
- appropriate use of constants;
- appropriate decomposition;
- clear organization (eg, grouped header files, struct definitions, function prototypes);
- well-named variables, functions, and other identifiers;
- verification of command-line arguments, if applicable;
- checking and appropriate handling of all return values;
- consistent style (capitalization, indenting, etc).

Submission Process

Your submission files should be in a git repository on basin in this directory:

```
/home/<username>/<secret>/assignment7
```

Email me when you want feedback, with a description of the type of feedback you would like.

Last modified: 11/16/2022 21:02:25