**ECE 252: Systems Programming and Concurrency** Spring 2019

# Lecture 9 — Socket To Me

*Jeff Zarnett* *2018-09-06*

## In Class Exercise: So Here's My IP / So Call Me Maybe?

**Background.** A very large number of programs will in some way communicate over the network with a remote server. Whether this is downloading aa file, handing off work from your phone to a more powerful machine somewhere, querying data from a database, or simply checking for updates, programs do this all the time. Now that we have had an introduction to sockets we will use them for a simple exercise where you send data and receive an answer. The overall description is simple, but there are quite a few steps involved.

**Primary Objective.** The primary objective of this exercise is to practice communicating with a remote server using socket communication in C.

**Secondary Objective(s).** You will also gain some more experience using the toolchain (gcc mostly) that should have been introduced in the labs for this course. This may also be your first introduction to git and gitlab.

**Starter Code.** The starter code can be found at `https://git.uwaterloo.ca/ece252/ece252-e1` – fork this repository to your own space. If you have not set up a gitlab account yet, you should do this now. Set permissions for your new repository to be private, but add the group for the course staff with read access so your code can be evaluated.

**Submitting Your Code.** When you're done, use the command `git commit -a` to add all files to your commit and enter a commit message. Make sure the `output.txt` file is included and has the answer you got back from the server. Then `git push` to upload your changes to your repository. You can commit and push as many times as you like; we'll look at whatever was last pushed. And check that you gave the course staff permissions!

**Grading.** Binary grading: 1 if you have made a meaningful attempt at implementing the code; 0 otherwise.

**Description of Behaviour.** The goal is to implement the program so that the following behaviour occurs:

- Your program should take in the IP of the remote server as the first argument. So if the server IP to use is `127.0.0.1` then the program is invoked like `./socket-client 127.0.0.1`.
- Establish a TCP/stream connection to the server specified (actual IP will be given in the class because these things can and do change with time!)
- Send your 8-digit student number to the server as a character array (`char[]` or `char*`) (string).
- After you send your number, try to receive a 64 character response from the server. If your upload was unsuccessful (i.e., you didn't send a valid student number) the remote server will close the connection without sending a response (and this indicates an error).
- Close the connection after you got the response.
- The answer you get back from the server should be written to the file called `output.txt`. The output doesn't look particularly human readable, but that's okay; the course staff know how to interpret it.
- Your program should not leak memory; be sure to destroy/deallocate anything initialized/allocated.

**Hints & Debugging Guidance.** All the steps you need to take to set up the communication, and send/receive the answer have been described in the previous lectures on sockets and network communication. If you read through those, it gives you the step by step as well as the syntax. You might also find the textbook([SR13]) helpful.

Some general guidance is below. If you're having trouble, try checking these things, and it may resolve your problem. If you're still stuck you can ask a neighbour or the course staff.

- Check the documentation for how functions work if you are unfamiliar with them (google is your friend!)
- Check return values (and possibly `errno`) for network functions if things are going wrong.
- Have you initialized all variables? It is easy to forget; `malloc` does not initialize the value...
- Pay attention to compiler warnings. Warnings can sometimes be ignored, but may be a hint about a semantic error that you will want to resolve. If the compiler is trying to tell you something, listen.
- You need a buffer for receiving the data, and the receive function tells you how many bytes the server sent.
- Don't forget null terminators for strings! Especially one you received via the socket.
- Is there a missing or extra $*$ (dereference) on a pointer somewhere?
- Does every memory allocation have a matching deallocation?
- It may be helpful to put `printf()` statements to follow along what the program is doing and it may help you narrow down where the issue is.
- Don't be shy about asking for help; the TAs and instructor are here to help you get it done and will help you as much as is reasonable.

## References

[SR13] W. Richard Stevens and Stephen A. Rago. *Advanced Programming in the UNIX Environment, Third Edition*. Addison-Wesley, 2013.