ECE 252: Systems Programming and Concurrency Spring 2019

Lecture 9 — Rogue One: A Star Wars Story

Jeff Zarnett *2018-10-26*

In Class Exercise: Scarif Don't Like It

Background. You are Jyn Erso, a traitor to His Majesty the Emperor, a member of the vile rebellion. You are trapped on the planet Scarif, having stolen the plans for the Death Star, and you plan to transmit them to the rebel fleet above. Time is short, because Lord Vader is on his way... and you know how he deals with traitors. You need to upload the plans, in the hopes that the rebellion can analyze them and possibly find a weakness. Upon transmission of the plans you'll receive an answer from the command ship indicating whether or not the transmission was successful.

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. See the guide for the in-class exercises for first time setup and guidance!

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 rebel command ship (remote server) as the first argument. So if the IP to use is `127.0.0.1` then the program is invoked like `./rogue-one 127.0.0.1`.
- Establish a TCP/stream connection to the command ship (actual IP will be given in the class because these things can and do change with time!). This is the client-side, so you need to get the address information (`getaddrinfo()`), create a socket (`socket()`), and then `connect()` to the remote server.
- Once connected, transmit the plans for the Death Star! The plans are loaded for you in the `load_plans` function (which loads the file into memory for you – remember to deallocate the data when done). The plans are large, so you probably cannot send it all in one call to `send`; so check the results and send repeatedly as needed.
- After you make your transmission, try to receive (using `recv()`) a response that could be up to 64 characters long. If your upload was unsuccessful you will get an error of some sort, otherwise you will get back a message that indicates success. You should print this message so you can see what it says. If it says "Success", it worked!
- Close the connection.
- 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.