Post-project Writeup - FDR

James Viner

Project Summary

The objective was to create a program that hosts a service on several ports that when sent a string in the format of the letter 'f', 'd', or 'r' followed by a number in the appropriate format, would send back the result of the requested operation in hexadecimal. The available operations are to be the Fibonacci sequence at the given number, the hexadecimal representation of the given decimal number, or the hexadecimal representation of the given Roman numeral.

Challenges

I think the biggest hurdle for this project was spending several hours attempting to write my own bigint library in order to solve the 8-point additional feature, which seemed like the natural approach given that the number that would need to be contained is several bits too large to fit inside of a 64-bit unsigned integer. I went back and forth on this one, first attempting to design it like the fibonacci program, just creating a number as two 64-bit unsigned integers to store its upper and lower halves, but then I realized that in order for it to be properly usable I'd also need to figure out how to perform the basic mathematical operations on each number, and that was surprisingly easier in Assembly than it is in C. At some point, I relented and looked online for another approach to the specific problem at hand and went with a solution that solves the immediate issue of needing arbitrary conversion of a decimal number in ASCII string format to a hexadecimal number. It ended up being a 50-line solution and works for any possible input.

This next one is also a bit smaller, but I didn't actually realize that my syntax for the netcat option to wait only for a certain amount of time before closing the connection (mandatory for my testing script) was wrong. On our VM's distro, it's '-q' instead of '-w' as it shows in the project handout and I spend more time than I'm proud of trying to figure out what was wrong with my program when it was just using the wrong option.

Successes

It's a pretty simple implementation, but I'm quite proud of my solution to the Roman numeral conversion. I think it's clean, readable, and may not be the absolute shortest way to write it, but it walks through the problem in the same way that you would while trying to read it, and I think that's neat. Bash testing was also fun this time around, since I had to re-learn the syntax for how to run a job in the background so that I could host the server, bringing it down occasionally to change options, and send it data. It doesn't run super quickly given that it has to wait a reasonable amount of time before closing each connection and when starting up the server to not run into issues, but it does work, and that's the important bit.

Lessons Learned

Given the opportunity, I should just use one of the many public bigint libraries out on the web instead of implementing my own. I know that for this particular course that isn't allowed, or at least not for this project, but it really comes

back to the universal realization I've been having about programming since the start of the course: I'm probably not the only one to have had any particular problem and there is probably already a library that does exactly what I need. Additionally, importing an assembly program for use in C isn't actually as difficult as it originally seemed. Sure, the function needs to be set up to properly receive its arguments from the caller, so a particularly simple program might need to be changed around a bit, but it's really not more complicated than having the compiler include it when building out object files and declaring it somewhere in the C code for later use.