Reliable UDP File Transfer

Jarred Parr, Kyle Flynn

October 2018

1 Introduction

The process of this project was nothing short of grueling. Kyle and I have some network programming experience, but the C languages tendency to be an unrelenteing beast at all times makes for a unquestionably hopeless tirade of misery whenver things get to be extremely granular. To assist in shrinking the frequency of these types of occurances, C was chosen for this project. The idea is that the more exposure we have will give us the best results. Also the sockets library in python is woefully slow and Java is. . . well. . . Java.

2 Design

The project was designed with a partially functional programming mindset. The idea was to have the code be robust and remove the chances of side effects. You will see attempts at this in the uses of abstraction for the more menial sections of code. This is to avoid dumb problems relating to debugging the more mundane sections of the code. This also allows for single points of failure. Side effects in code give way to unforseen bugs and other external factors making a mess of the place. As a result, we sought to remove as much of that, within reason, as possible. We used two files: *server.c* and *client.c* to house our code. We tossed around the idea of constructing library functions for repeated tasks, but with time being of the essence, we opted to wait until the next project to attempt those. As a result there is a bit of code duplication, but it was largely due to the lack of available time since a large portion of the project was solved relatively close to the due date.

Outside of that, things remained within a rather simple and straightforward design. We aren't experts, here. At least not yet.

3 Challenges

Most of the challenges were largely dealing with memory allocation, file navigation, and reliably sending data over the wire. Error checking was a bit of a nightmare in most cases and required a lot more thought than was initially anticipated. To solve the issues a great deal of pair programming was done. Testing code, conversing about ways to fix problems, and implementing solutions to varying degrees of success allowed us to keep a strong cohesion as groupmates and determine main pain points that we could work on when separated. This resulted in us largely being on the same page, and each github commit resulting in productive, non-duplicated changes occuring.

4 Compiling

The code was compiled with gcc 8.2.1 on arch linux and can also be compiled via the provided make file. Compilation instructions via the provided file are shown below, in order to get the intended results of the code, please place each file into a folder client/ and server/ for client.c and server.c respectively, the makefile will not work otherwise and file transfers will not be able to share the same name (which was not accounted for in this project)

- Segment the pieces of code into their own folders as stated above
- Run make all or simply make to get the executable
- Run ./server/s port_number and then ./client/c host port_number
- Follow the on-screen instructions to begin the file transfer from server to client