Daniel DeRuiter

Professor Andrew Kalafut

CIS 457

3 October 2016

Project 1: TCP File Transfer

The goal of this project was to allow files to be transferred over a TCP socket, from a server to a client. Below is a description of how I achieved this with both the client and server sides.

To begin, I will discuss the design choices I made when creating the ‘Client’ program. The client must first enter the port that the server is on, following that the user will be prompted to enter an IP address, which is the IP address of the server. These will both be used to create the socket that will connect to the server, and will eventually be used to transfer data between the two. After this, I fill in the sockaddr\_in struct with the appropriate information for creating a TCP socket, and then attempt to connect to the server. After a successful connection, we begin the file transfer process.

The first thing that should be pointed out is that the rest of the ‘Client’ program is thrown in to an infinite loop. This is done so that the client can request multiple files over the same connection, and can quit this loop by entering the letter ‘q’. Once the loop starts, the user is prompted to enter a file name. Immediately after, the file is sent over the socket to the server. The client will then wait until the server sends back the size of the file that the client requested, this size is needed so that the client will know how much space to allocate and to know how much data will be read from the socket. This is probably the most important aspect of receiving the data, the client needs to know how much of the socket it needs to read, otherwise it could be reading incorrect data.

After receiving the size, the client will open a file with the same name (essentially create/overwrite the file locally) and prepare to receive the file contents as an array of bytes. Using an array of bytes will allow any kind of file to be transferred over the socket. An array will be allocated to the size of the file, and will then read from the socket the file contents. After reading all the bytes, the client writes the bytes to the file that was created, and the loop starts over.

For the server side, the user is first prompted for the port that the server will be listening to. After the socket is set up, the server is thrown in an infinite loop. Inside the while the blocking ‘accept’ call halts execution until a client connects. After a client connects, a new child process will be spawned to handle the processing for this client. I chose to use a multi-process program because I can handle as many clients as needed, and it didn’t require as much code or work as using threads. Once the child process hits the if condition, the child will continuously receive file names from the user. This is done by putting the ‘recv’ call in the while, so each loop iteration is a file request from the client. When ‘recv’ is less than 0, the child will then know that the client has closed the connection, so the loop stops, and we close the client socket. Inside the while, the child looks for the file that the client has requested. Then, the child will determine the size of the file, and send it to the client. Next the file is read and stored into an array of bytes, which will be written to the socket, and received by the client. After the transfer has complete, the file is closed and the loop restarts.