Sidharth Gilela
sgilela@ucsc.edu
Assignment 1
CMPE 156

Design Document

Goal:

The objective of this project is to implement a client server model such that the client and server can communicate with one another. A goal of this assignment was to help create a solid understanding of what exactly occurs at the socket level during this interaction. I used various system calls to allow this and had to deal with various errors. It was a great assignment to have as an introduction to network programming. I created wrapper functions for these system calls, used Wireshark to see the packet traversal, and learned gdb for debugging.

Run and Compile:

To compile the client and server run these commands from the src folder:

```
make client
make server
```

To run the test script run this command from the src folder:

```
make test
```

To remove the executables run this command from the src folder:

```
make clean
```

Design:

In order to describe my design and how the messaging between the server and client is done, I will briefly go over a simulation between the client and server. Along the process, I will explain error handling scenarios and more. The first thing that happens is that the server assigns a socket on its side and returns a number below 0 if failing to do so. An error is shown on the screen if there is an error. The server then sets up its information such as its port number, and its family which is AF_INET. It will then listen on the socket that it has assigned itself for an infinite time waiting for a client to connect to it. Now the client can start to run which assigns itself a socket as well. It also holds attributes such as the port number on the server to connect to and more. Now, after this initialization and starting phase, the client will send a message to the server asking for a connection to be made. The client does this with the connect system call. As a brief mention, every time any system call is made, I do a check to make sure that the returned integer is not below 0. If it is below 0, an error is printed and the client terminates, ending the connection. The only exception would be in the very beginning where I have a check on both the client and server side, making sure that the number of arguments to run the program is correct. Now, to continue where I left off, the client send a SYN message to connect with the server. The server replies with a SYN and ACK and then the client responds with an ACK possibly piggybacked with some data. These SYN and ACK are flags set that can be seen during packet traversal on Wireshark if wanted. Unfortunately, I actually have not implemented a security measure to the server to let the server know that it is a client that wants to connect with it because I don't think its mentioned in detail to execute on the assignment. I know that in order for this to happen the client will need to send some sort of message like "I am a client" and the server responds with "Server is ok with you". This is to ensure security that the client is really who he/she is and it's not some unknown source.

When the server gets the client’s connection, it actually then runs the accept system call which assigns a different socket on the server side for the client and server to communicate. The reason for doing so is because then that single port would never get flooded with constant requests in the case of multiple clients trying to connect with it. After the connection has been made, the client sends a command to the server that the server can execute. I have written my code such that the command cannot be more than 1024 bytes and my reason for doing so is because the maximum length of a Linux command in general is around 239 bytes. Thus, I don’t have a while loop that continually reads in input from the client and sends it to the server. The server then reads in the command and then runs popen on it which will execute the Linux command. The server then sends the data after execution in a loop because there could be more than 4096 bytes of data which is the size of my send buffer on the server side. It will continually send 4KB of data to the client until there is no more information to read from popen. Thus the client has a 4KB receive buffer that will continually receive incoming data and display the results. Both the receive buffer on the client side and the send buffer on the server side are of length 4KB. They could honestly be any size but they both have to be the same. The reason being is because there would be a case where the client actually receives 4KB of data all at once and it put its limit as below 4KB. This could cause a segmentation fault for accessing memory that is not allowed. After the command is done sending on the server side, a new connection is made for the next incoming command. This new connection includes the accept system call and the rest as explained above. The client reuses the same socket for every command that it wants to send over. At the very end, the client will send an exit message to the server. When this message is received by the server, the server closes the socket that is assigned by accept so that it can still listen for an infinite time waiting for more connections. The server sends an exit message back to the client for the client to close its socket and then the client closes its socket. This is the moment when the server reads in zero bytes so the read value on the server will be 0. When this value is 0, it will then close that socket that I have just mentioned above.

Notes:

I have not included any unit tests and the teacher has told me that it was ok not to include any unit tests for just this program since it is the first one.