Stephen Longofono

EECS 678

Lab 10 Write-up

1. Briefly describe the design of the program. How does your program control when the client runs and when the server runs?

   The program begins by setting up a socket address struct to hold the type of socket and the address used to refer to it. This is used to generate a handshake socket. The handshake socket is then set to listen on the specified address, which allows a client socket to request a connection. When the client requests to connect to the address, the connection is passed off to a new socket, which is used for communication thereafter. In a more complete program, the handshake socket would then resume listening for other clients.

   The program controls when the server runs and when the client runs by inserting a 1 second sleep between when the server is started and when the client is started. Within the server and client programs, the socket system call maintains the duplex connection for the life of the process.

2. What is the purpose of the handshake socket? Why not have the server create and bind session sockets that clients may connect to directly?

   The handshake socket allows a server to service multiple clients at the same time. Without a handshake socket, whenever the server was reading or writing data to a client, all other clients would be unable to connect to the server. When a client tries to connect to the address, the listening handshake socket can respond and initiate the process of setting up a new socket for communication with that client.

3. For the simple / client server program, we chose to use sockets instead of pipes to send messages between the client and server. Why are sockets preferred over pipes for this program? Give at least two reasons.

   Sockets are preferable over pipes because they are designed for two-way communication, and they allow connections which extend outside of the operating system. Pipes can be set up to do two-way communication, but it is cumbersome and lacks the useful features of a socket system (handshake sockets, parity checking, and packetizing for example). Sockets also allow sophisticated communication among computers and networks, which is difficult if not impossible in many situations with pipes (TCP and UDP for example). I found a few examples of using named pipes or linking sockets to pipes for communication outside of the host machine, but they were exceptional cases.