**Socket Programming Notes – CPE 464**

* Creating a Socket/Opening a file
  + You need to check **all** your system calls return values!!!
    - For example: **< 0** means **error**
  + Checking for errors on system calls (< 0) will save you **hours of wasted debugging time!!!**
  + Perror() is your friend!!!
  + However, if there’s an error and you don’t give it enough buffer space, or a bad something or other and it happens in the kernel, the kernel’s not going to segmentation fault because then you would just be freezing your machine every time you screwed up a system call. Therefore, what is going to happen is the system call is going to stop and it’s going to return -1 and if you’re checking for that, then an error is occurring over here, and you have no idea that it occurred. So, it is completely different than making normal calls. It is very important that you check your system calls for errors because if you don’t, you may spend hours and hours debugging it because it is not working in the kernel, and you don’t know because you are ignoring the return value. Every time you make a system call, you just check if it is less than 0 on the return call, and if it is you print out a perror().
  + When you make a system call to print something to standard or to read something from standard in, it’s going to be a system call and it’s going to go access some memory in the kernel that contains information about “standard in”.
  + 0 🡪 stdin (standard-in)
  + 1 🡪 stdout (standard-out)
  + 2 🡪 stderr (standard error)
  + Therefore, when you open a file, the lowest file descriptor you going to get is 3.
* Types of Sockets – TCP and UDP
  + There are 2 types of sockets that we can create:
    - Transmission Control Protocol (TCP)
    - User Datagram Protocol (UDP)
* **Naming a Socket** – Uniquely identify the Server End Point
  + Since File Descriptor 0 is STDIN – this is a bad example.
  + Use File Descriptor 1 in this discussion (STDOUT)
  + File Descriptors are only known and accessible by one process. They are internal to the process.
  + A **port number** is a number that is maintained by the kernel that’s only given to one connection.
  + **Port Number** – Defines the communication endpoint (e.g., your web server).
    - For example: If I open a socket and get a port number, I get that port number and nobody else on the machine can have it. And if you open a socket and then you ask for a port number and you get a port number at 1) it won’t be the one I have and 2) it’ll be unique on the machine for you. That way, the kernel is what maintains the port numbers and port numbers are maintained by the kernel. The kernel then knows where to send it to. And we are going to amp the port number to a socket number but the socket number’s local the port the port numbers global. We are still going to talk to read and write to the socket like we do any file descriptor but we have to call it bind() the socket to a port number so that it has that end point of the connection for the rest of the world.
  + What info is needed for a client to **Identify** a server?
    - Naming the socket:
      * Protocol Family (AF\_INET6)
      * IP address of the server.
      * Port number the server is using.
  + Every packet that goes out, has the client’s IP and the server’s IP, the client’s port number and the server’s port number. And that uniquely identifies a complete end-to-end connection, and I can use that to talk both ways. Therefore, all I need is those four pieces of information for sort of the two-way communication.
* Client – connect() System Call
  + Calling getIPAddress6() (so DNS) causes communication to happen with external DNS servers.
  + So getIPAddress6() could take a few seconds to return