CPE 464 Lab – Socket API and using poll()

Due:

- Lab worksheet (pdf) – see canvas for due date.
- The lab programming assignment is in a different document – **do this worksheet first**.

Name: August Hogen-Esch

Each student must type up their own answers. While you should work as a group, you are required to type up your worksheet and turn in a copy of your work.

I. **If you worked in a group, please include your group members' names.**

Other Group Members' Names (max of 4 to a group)

II. **Socket API** (Use my sockets lecture and Google to find answers to these questions)

This assumes you have watched the two videos on Canvas.

1. Write the C language command to create a TCP socket for IPv4/IPv6 family?

```
if( socketNumber = socket(AF_INET6, SOCK_STREAM, 0) < 0){
        perror("socket call");
        exit(EXIT_FALIURE); // typically -1
}
```

2. Regarding the bind() function
    a. What information is needed to **name** a socket?
        i. Protocol Family
        ii. IP Address of the server
        iii. Port number that the server is using
    b. Why do you only need to call bind() on the server? (so why do you need to call it on the server but not on the client)
        i. You need to bind the socket on the server side to explicitly give the socket a port number, in order to know which port the client should be trying to connect to
    c. How does the client get the server's port number?

        i. Typically as a runtime argument, as the server's port number must be established before attempting to connect. After calling bind(), you have to use getsockname() to return the port number, which can then be passed to the client.
    d. If we do not need to call bind() on the client, how does the client get assigned a port number? (this question is NOT about the server's port number but about the client getting its own port number)
        i. When you send on a socket without a port number, one is assigned by the kernel. We don't care about the client's port number, so we get our port number implicitly.
    e. How does the server get the client's port number?
        i. The server gets the client's port number when the client connects to it. IMplicitly, a port number is generated for each client that connects.

3. Regarding the accept() function
    a. When will the accept() function return (remember it's a blocking function)?
        - Accept is a blocking function, and waits for a connect()
    b. What does the accept() function return (besides -1 on error)?
        - Accept returns a new socket number for every client that connects

4. When using TCP on the **server** to communicate with clients there are at least two different sockets involved on the **server**. Explain:
    a. There is one 'main server socket' which is the first socket created and is used to set things up and call accept(). For each client that connects, including the first client, a new socket is created for that connection.

5. Regarding the poll() function (Google man poll):
    a. What does the poll() function do?
        - The poll function probes a socket (or set of sockets), and returns if that socket is ready to be read. Poll can be set to block indefinitely, for a certain amoutn of time, or return immediately.
    b. Name/explain the parameters passed into the poll() function.
        - THe three parameters of the poll call are the number of file descirptors, a set of sockets, and a timeout value.
    c. Regarding the timeout parameter passed to the poll() function
        ii. What value will cause the function to block indefinitely?
            i. -1
        iii. What value will cause the function to not block but just look at the socket(s) and return immediately?
            i. 0
        iv. How can the function be made to block for 3.5 seconds?
            i. 3500

6. How does the server (or client) know that the client (or server) has terminated (e.g. ^C, segfault) when using the recv() call?
    a. If you ever get a return value of 0 during a recv(), you know that the connection was closed.

7. Explain why we need to use the MSG_WAITALL flag on the recv() function call in program #2.

a. We need the msg_waitall flag so that recv will wait to receive the entire PDU. This way we can ensure that it will try to capture all the data being sent.

8. Regarding the code provided with the lab (it is the same code provided with programming assignment #2). The purpose of these questions is to help you understand the code I have provided.

   These questions all concern the file: **networks.c**

   a. Write the line of code (from network.c) that creates the server_socket:

```
mainServerSocket= socket(AF_INET6, SOCK_STREAM, 0);
    if(mainServerSocket < 0)
    {
        perror("socket call");
        exit(1);
    }
```

   b. What are the line numbers for the code that is used to name the server's socket?

      lines 43 to 53 build the structure and bind the server's socket, with error handling.

   c. What is the effect of the following:

      - What does the In6addr_any do (so what does bind() do)?

        (e.g. serverAddress.sin6_addr = in6addr_any;)

      it is a parameter used to indicate we just want to use any of the physical ip address the machine has, and is sued instead of hard coding Ip addresses.

      - If I use a port of 0 in my call to bind(), what does bind() do?

        e.g.

        serverPort = 0;

        serverAddress.sin6_port= htons(serverPort);

        Passing 0 as the port will tell bind() to choose a random available port. Even if you use htons(0), it will still be setting it as 0, and will be a randomly designated port number.

III. **Regarding TCP:**

There are several function calls that must be made on a client and server in order to utilize sockets for communications (all in networks.c except send()/recv() and close() which are in myClient.c and myServer.c).

1. In the table below, list the function names (see socket video slide 4) in the order which are needed to setup, use and teardown a connection using Stream Sockets (get this list of functions from the video slides on sockets).
2. In the table below, connect the functions that are part of the connection oriented part of TCP using arrows.
3. Looking at the network.c file provided with this lab put the line number of the call to that function in the table. (Don't worry about line numbers for send()/recv() and close() since these are called in myClient.c and myServer.c.)

Client Line #	**Functions called on the Client**	**Functions called on the Server**	Server Line #
110	socket()	mainSocket = socket()	36
		bind(mainSocket …)	49
		getsockname(mainSocket …)	56
		listen()	62
127	connect() →		
		clientSk = accept(mainSocket …) ←	82
	send() →		
		recv()	
		send() ←	
	recv()		
	close() →	close() ←	