Mary Madeline Nicole 109006205

1. What does INADDR_ANY mean? (10pts)

   INADDR_ANY is a constant in network programming that represents the IP address 0.0.0.0. Where it is used in server programming to indicate that the server should bind to all available network interfaces on the host machine. When a server socket is bound to INADDR_ANY, it can accept connections from clients on any IP address that the host machine has. It is more or less a special IP address that binds the socket to all available network interfaces in the machine. This is very useful when the IP address of the network interface used is unknown.

2. What's the difference between bind() and listen()? (10pts)

   A specific IP address and port number must be associated when creating a server socket, bind() binds the socket to the specified address and port. Once the socket is bound to an address and port, then by the listen() function, it can be set to listen for incoming connections. So bind() specifies the address that the server should listen on and listen() sets the maximum number of incoming connections that the server can handle. Both of these functions are used for accepting incoming connections from clients.

3. Usually, we set up the server's port and exclude the client's. Who determines the client's port and what's the benefit? (20pts)

   The client's port is usually assigned by the operating system. The OS assigns a unique port number to the client's socket, by assigning a unique port number to each client, the server can distinguish between multiple connections from the same client, this also allows multiple clients to connect to the same server using different ports..By assigning a unique port number to each client, the server can distinguish between multiple connections from the same client. For example, if a client opens two connections to the server, one for file transfer and one for video streaming, each connection will have its own unique port number, so their data is ensured to not be mixed up. This allows the server to keep track of the different types of connections and handle them separately. So, the benefit is that it allows efficient and reliable communication between multiple clients and a single server and it also prevents conflicts and confusion between different connections.

4. What is little endian and big endian? Why do most network byte order use big endian? (10pts)

Little endian and big endian refer to the way in which multi-byte data types are stored in memory. In little endian, the least significant byte is stored first, followed by the next least significant byte, and so on. This means that the least significant byte has the lowest memory address. In big endian, the most significant byte is stored first, followed by the next most significant byte, and so on. This means that the most significant byte has the lowest memory address.

Most network protocols use big endian byte order because it makes it easier to compare values between different systems, as they will all be using the same byte order.

For example, the integer value 0x12345678 would be stored in memory as:

Little Endian		**Big Endian**	
Address	**Byte value**	**Address**	**Byte value**
0x1000	0x78	0x1000	0x12
0x1001	0x56	0x1001	0x34
0x1002	0x34	0x1002	0x56
0x1003	0x12	0x1003	0x78

5. Why do we need a pseudoheader ? (10pts)

When data is sent over a network, a value called a checksum is calculated to check if any errors occurred during transmission. This value is calculated by adding up all the bytes in the packet and then taking the one's complement of the result. However, this method is not always reliable and can lead to false-positive results for some specific conditions.

To prevent such issues, a pseudoheader is added to the packet before calculating the checksum. This pseudoheader includes additional information such as the source and destination IP addresses, protocol number, and packet length. The checksum value is then calculated over the pseudoheader and the packet itself. which helps to detect errors that may have occurred during transmission and ensures that the receiver can verify that the packet was not tampered with during transmission.

Adding a pseudoheader to the checksum calculation also helps to ensure that the packet is being sent to the correct destination and prevents packets from being falsely accepted due to checksum collisions.

6. For the code below, what's difference between client_fd and socket_fd ? (10pts)
client_fd = accept(socket_fd, (struct sockaddr *)&clientAddr, (socklen_t*)&client_len);

The key difference between socket_fd and client_fd is that socket_fd is the socket descriptor for the server's listening socket, which is used to accept incoming connections from clients. When a client connects to the server, accept() function is called on socket_fd to create a new socket descriptor client_fd, which is specific to that connection. socket_fd remains in the listening mode even after a client has connected, so it can be used to accept more connections from other clients. On the other hand, client_fd is specific to a single connection and is used to send and receive data to and from that particular client. So, socket_fd is used to accept incoming connections from clients, while client_fd is used for a specific connection to send and receive data.

7. When using the send() function and recv() function, why do we not need the address? (10pts)

   When using send() and recv(), the address is not required because the socket is already connected to the remote address. When the socket is created and connected, the address was already specified. We only need to know which socket to send or receive data from, because when the send() function is called, the socket already knows the destination address and port number to send the data to. Also when the recv() function is called, the socket already knows from which address and port the data is coming from.

   The address would be needed when the socket is not yet connected, when connecting using connect(), the address specifies host and port number for connection. Once connected, the address is no longer needed for the send() and recv() function.

8. Write about what you have learned from Lab 2. (20pts)

   I learned how to do TCP socket programming and create the TCP header. I think that TASK 1 of connecting the server and client with TCP socket was quite simple and easy to understand, therefore I was able to successfully send a message. Then, creating the header without checksum using l4info was a little complicated, as I have to be precise and make sure that the headers are put accordingly to create the right result. The last task, task 3 was the biggest challenge of all. Even though I did create the checksum function, it keeps on getting errors and computing the wrong checksum. The checksum I think is the most tricky part. First separating the ip address, and changing them into integers, and also with the headers and pseudo header. I learned to be thorough and to be very careful in calculating as the simplest mistake could result in the wrong answer. But overall, I really think it was a great experience to be able to create TCP sockets and calculating the checksum.