CS 472 Assignment 4 Written Part

1. Go through the code in du-proto.c and make sure you understand it. After that, briefly place a comment block at the top of each function describing at a high level what that function does to enable the protocol.

Each function contains a comment block with a description of how the function works and operates. For this assignment, I modified 4 functions: dpsend(), dprecv(), dpsenddgram(), and dprecvdgram(). The comment block for these functions reflects the modifications I made. There is also a comment block in du-ftp.h describing the FTP PDU we were instructed to create, which also includes some notes for the grader.

2. Three sub-layers were used for various parts of the transport model. If you look at dpsend() and dprecv(), each one of these is supported by two additional helper functions. What are the specific responsibilities of these layers? Do you think this is a good design? If so, why. If not, how can it be improved?

The datagram helper functions are where the ACK happens and PDUs are built and handled. The raw helper functions are internal functions at the socket level that allow for ACK at a low level. This simulates a real protocol, where each helper function has its own responsibilities. The main functions are catalysts for these helper functions — each one calls the datagram helper functions, which then call the raw helper functions. This main function is what is called by the application layer when dealing with transfering the file contents.This separation allows for a clear designation of what is going on and ensures certain operations and responsibilities are encapsulated within their own functions, not muddied by the functionality of the others. I believe this is a good design because, while it might require more understanding of how everything works in unison, it really drives home the idea that each function should have its own specific set of responsibilities. I also feel that this design yields one of the key aspects of developing software — high cohesion and low coupling. Everything within each function is necessary to one another and relies on this data, which allows each function in respect to the others to not need to pass a lot of data for it to operate appropriately.

3. Describe how sequence numbers are used in the du-proto? Why do you think we update the sequence number for things that must be acknowledged (aka ACK response)?

Sequence numbers are used in the protocol to identify how many bytes are sent, incremented by 1 to identify an ACK of that send. The sequence number is incremented for an ACK because this will identify that something is being sent that requires an ACK. If an ACK was not necessary or desired, the sequence number wouldn't change — it would remain the number of bytes sent. This increased sequence number system is essential in verifying that the data was received.

4. To keep things as simple as possible, the du-proto protocol requires that every send be ACKd before the next send is allowed. Can you think of at least one example of a limitation of this approach vs traditional TCP? Any insight into how this also simplified the implementation of the du-proto protocol?

In TCP, multiple sends can be done before an ACK is received — an ACK can be done even just one time regardless of the number of sends. In this protocol, we require every send and receive to be ACKd, meaning that there is more overhead. This is especially crucial depending on the size of the datagram in which you can send and the file being transferred. For example, if your maximum datagram size is 512 bytes and you are transferring a 500 megabyte file, then over a million ACKs will be done. However, to achieve the TCP ACK behavior in this protocol, some sort of threading mechanism would have to be implemented. By ensuring that an ACK is received before sending, a simpler structure is created, albeit slower compared to TCP or threading.

5. Briefly describe some of the differences associated with setting up and managing UDP sockets as compared to TCP sockets.

There are a few differences between TCP and UDP sockets which revolve around the fact that TCP is connection-oriented and UDP is connectionless. The difference between TCP and UDP sockets on the client side is that a TCP socket uses connect() to establish a connection to the server. Since UDP is connectionless, a socket of this type does not need to use connect(). On the server side, TCP sockets use listen() and accept() to listen for connections and accept said connections. In the same vein, since UDP is connectionless, this does not need to happen. Furthermore, UDP sockets do not need to utilize close(), as a connectionless protocol does not have any requirements for accepting, maintaining, and closing connections. The last difference is how TCP sockets more often use send() and recv(), while it is preferred to use sendto() and recvfrom() with UDP sockets. Since UDP does not manage connections like TCP does, these unique send and receive functions allow for the IP and port to be specified for each outgoing and incoming packet.