MP1 Questions

Question 1: Why do the pairs of processes not interfere when using same port?

The reason that both server and listener can run on the same port on the same machine without interference is the type of socket that is created. There cannot be two stream or two datagram sockets on the same port, and an attempt to run two instances of listener or server will result in a binding error. Server creates a stream socket for TCP communication and listener creates a datagram socket for UDP communication. To determine this, I ran `lsof -i:4950` to see what was using port 4950, and changed the datagram socket to be a stream socket in `listener.c`. After this change, listener and server could no longer be run together because of a binding error.

Question 2: Print a copy of the information the server receives

The following informatin is from the `wget` command instead of the web browser:

server: waiting for connections... server: got connection from 130.127.206.77 server: received message: GET /home/bschlue/6380/mp1/dummy HTTP/1.1 User-Agent: Wget/1.20.3 (linux-gnu) Accept: */*Accept-Encoding: identity Host: apollo09.ces.clemson.edu:4950 Connection: Keep-Alive

START_LINE: GET /home/bschlue/6380/mp1/dummy HTTP/1.1 MESSAGE_HEADER: Host: apollo09.ces.clemson.edu:4950

Part C

To achieve the functionality outlined in the instructions, several changes had to be made to the `client.c` and `server.c` files from Beej's Guide to Network Programming.

The changes to the server included creating a table with id numbers and strings, modifying the call to `recv` to store the payload in a variable representing ID number, and adding a `send` call to send the appropriate message back to the client, or an error message if the ID number does not correspond to a table entry. The server protects itself from receiving malicious long input by only reading in a single integer in it's receive function. Therefore, if a malicous client sends a huge amount of data, it will be cut off after only 4 bytes.

The client was modified to take an argument of an ID number, which was then validated. The data sent to the server was updated to be the number passed in. When the client receives a response from the server, it is stored in a buffer of size equal to the max message size that the server can send to prevent any data from being lost, which is configured to be 128 bytes. The client protects itself from malicious server's in a similar way that the server protects itself, which is by limiting the amount of data read in to a static 128 bytes.