Isaac Boaz

October 1, 2024

Lab 1: Socket API

For the questions below, you will need to start with and then modify the `demo_client.c` and `demo_server.c` files linked in the assignment posting.

Note: The term ‘localhost‘ refers to the loopback network interface of the machine you are working on, and the IP address ‘127.0.0.1‘ is often used to refer to it. When running the client, you can use either ‘localhost‘ or ‘127.0.0.1‘ as the server address to test communication with a server running on the same machine.

Some additional hints/instructions:

- Your answers can be a combination of written explanations and command line output. Please use a monospaced font for any command line output to clearly distinguish it.
- Sometimes, the answer to “what happens if... ” could simply be “it breaks, showing the error message ....” (Don’t spend too much time trying to force something to work that will not work.)
- Hint: For some of these questions, it may be useful to have one process pause while the other performs an action; consider using the `sleep` command.

Q1: (2 pts) **Port Questions**

(a) What happens when you run `demo_server` using a port in the range of 1 through 1023? Explain why this occurs.

(b) What happens when you use a port in the range of 1024 through 65535? Why?

(c) *Optional challenge question (time permitting):* What local port number is `sd2` (in the server) bound to? (Hint: Use `getsockname`). Is this what you expected?

Q2: (2 pts) **recv Questions**

(a) What does `recv` return when there is still data in the OS buffer, but the other side has closed the socket? To find out, modify your server to send a large amount of data (greater than the client buffer size) and then close the connection immediately. In the client code, add a `sleep` before calling `recv`. What do you observe when `recv` is called under these conditions?

(b) What does `recv` return when the OS buffer is empty and the other side has closed the socket?

Q3: (2 pts) **listen Questions**

(a) What happens if you comment out the call to `listen` in `demo_server.c`, then compile and run?

(b) If you set the queue size for `listen` to a small value (e.g., 2), does it actually limit the queue length to 2? How can you test this?

Q4: (2 pts) **send Questions**

(a) What happens when you change your server's `send` call to pass the address of a `uint32_t` variable instead of `buf` and use `sizeof(uint32_t)` instead of `strlen(buf)`- without modifying the client's `recv` call?

(b) What happens if you make the above change to `send`, and also modify your client to perform a single `recv` call that receives into a `uint32_t` variable of the appropriate size? Is the variable successfully transmitted without using any character arrays?

(c) What are `htonl` and `ntohl`? What purpose do they serve in socket programming?

(d) When would you need to use `htonl` and `ntohl` while transmitting `uint32_t` variables (i.e., before or after which actions)?

(e) Modify your client's `recv` call to use the `MSG_WAITALL` flag, and remove any repeated calls to `recv`. In the server, change the `send` function to send partial messages with a delay between each chunk. Test how the client receives data with and without the `MSG_WAITALL` flag. What differences do you notice?

Q5: (2 pts) **Socket Address Structure Questions**

(a) What are the fields of the `sockaddr_in` structure? Describe their types, names, and purposes.

(b) What are the fields of the `in_addr` structure?

(c) What does the special value `INADDR_ANY` mean, and when would you use it?

Q6: (2 pts) **Address Conversion Questions**

(a) Use `inet_aton` to convert a decimal-dotted string IP address to an `in_addr`.

(b) What does `inet_aton` return when the IP address is valid?

(c) What does `inet_aton` return when the IP address isn't valid (e.g., "905.0.0.1")?

(d) Modify your client code (`demo_client.c`) to convert the `h_addr` field from `gethostbyname` into a decimal-dotted string format. First, declare a character array of size `INET_ADDRSTRLEN` to store the converted IP address. Then, after calling `gethostbyname`, use `inet_ntop(AF_IN ptrh->h_addr, buffer, INET_ADDRSTRLEN)` to convert the binary IP address to its string representation. Finally, add a `printf` statement to print the resulting IP address string before establishing the socket connection.

(e) What is printed when you pass host names like "`google.com`" or "`linux.cs.wwu.edu`"? Does the output make sense?

(f) What happens when you pass an invalid host name like "`cf416-99.cs.wwu.edu`"? Why does this happen?