CSCI 4210 — Operating Systems
Lecture Exercise 4 (document version 1.1)
Network Programming and UDP (SOLUTIONS)

- This lecture exercise is due by 11:59PM ET on Wednesday, April 28, 2021
- This lecture exercise consists of practice problems and problems to be handed in for a grade; graded problems are to be done individually, so **do not share your work on graded problems with anyone else**
- For all lecture exercise problems, take the time to work through the corresponding video lecture(s) to practice, learn, and master the material; while the problems posed here are usually not exceedingly difficult, they are important to understand before attempting to solve the more extensive homeworks in this course
- As with the homeworks, you **must** use C for this assignment, and all submitted code **must** successfully compile via `gcc` with no warning messages when the `-Wall` (i.e., warn all) compiler option is used; we will also use `-Werror`, which will treat all warnings as critical errors
- All submitted code **must** successfully compile and run on Submitty, which uses Ubuntu v18.04.5 LTS and `gcc` version 7.5.0 (`Ubuntu 7.5.0-3ubuntu1~18.04`)

Practice problems

Work through the practice problems below, but do not submit solutions to these problems. Feel free to post questions, comments, and answers in our Discussion Forum.

1. Why are protocols so important to any form of network communication?

   SOLUTION: Each layer of the OSI Reference Model logically communicates with the same layer at the other endpoint. If a protocol is not clearly defined, an endpoint will receive data it does not know what to do with. Further, if a protocol has bugs in it, there may be cases where it is impossible to interpret data received at the other endpoint.

2. Why does the OSI Reference Model used a layered stack approach? What are the benefits of using such an architecture?

   SOLUTION: In general, using a layered stack approach keeps each layer focused on a specific set of communication requirements.

3. In `udp-server.c`, what would happen if a datagram with 16,384 bytes of data was received? More specifically, what would the server output to the terminal?

   Also, what would the data field of the response datagram sent by the `sendto()` system call contain? Show the bytes exactly.

   And finally, describe what data is lost, if any.

   SOLUTION: If the given `udp-server.c` was running and a datagram with 16,384 bytes was received, only `MAXBUFFER` bytes would be read, i.e., 8,192 bytes would be read, while the remaining bytes (another 8,192 bytes) would be discarded.

   The data field of the response UDP datagram sent by the `sendto()` call would be the first four bytes of the 16,384 bytes received followed by a newline character.

4. What is the maximum size of the data field of a UDP datagram? In other words, what is the largest possible payload a UDP datagram could transport?

   SOLUTION: From the UDP datagram structure given in the lecture notes for 4/20, the Length field is a 16-bit field that specifies the number of bytes in the entire UDP datagram. Therefore, we start with a maximum of 65,535 bytes, but subtract the UDP datagram header fields (i.e., eight bytes) to get to 65,527 bytes; however, we must also consider the IP datagram structure!

   For this, the Fragment Length field is also a 16-bit field that specifies the number of bytes in the entire IP datagram. An IP header has a minimum of 20 bytes, so we are down to 65,507 bytes as a maximum data payload for a UDP datagram.

   (This question is beyond the scope of this course.)

5. What class network is `usps.com`? As a hint, use the `host` or `ping` (or another) terminal command to help determine this.

   SOLUTION: The `usps.com` network is a class A network, determined as follows:

   (a) First, we need to determine the IP address of the `usps.com` hostname; to do so, try `ping usps.com` or `host usps.com`

   (b) Given the IP address (`56.0.134.100`), write the first octet in binary, i.e., `00111000`

   (c) Match the first few bit(s) to the DECODING IP ADDRESSES slide from 4/20; in this case, the first bit is `0`, so we have a class A network

Graded problems

Complete the problems below and submit via Submitty for a grade. Please do not post any answers to these questions. All work on these problems is to be your own.

SOLUTION: See posted code solutions on Submitty.

1. Copy the given `udp-server.c` code to a separate `lecex4-q1.c` file.

   Modify this program as follows:

   (a) The use of `MAXBUFFER` has a buffer overflow error. Fix this error by modifying one of the arguments to the `recvfrom()` call.

   (b) Add support for an optional command-line argument (i.e., `argv[1]`) that indicates the specific port number to bind the UDP socket to. If `argv[1]` is present and would be a valid port number, use this value as the port number in the `bind()` call.

   (c) Change the application protocol to send the last 16 bytes of the received data (instead of the first 4 bytes). Do not send any additional characters, such as a newline. If there are less than 16 bytes, simply echo back all of the received data.

2. After completing Q1 above, copy your `lecex4-q1.c` code to a separate `lecex4-q2.c` file.

   Modify this program as follows:

   (a) Set `MAXBUFFER` to 512.

   (b) Change the application protocol to count the number of `'G'` characters in the received data. Send this 4-byte `int` value in the response datagram. For data marshalling, be sure to use the `htonl()` library function. This will ensure that the data is correctly interpreted by the remote side.

Additional debugging hint

(v1.1) In your server code, output to `stdout` is block-buffered when run on Submitty. To ensure your server produces output that you can view, add `setvbuf()` as the first line of your `main()` code as follows:

```
setvbuf( stdout, NULL, _IONBF, 0 );
```

Note that your server output is **not** graded via Submitty, but this might help you debug any problems you face.