UDP Code Analysis Lab

(Rev. 2 Mar 2016)

In this lab you will analyze and modify code that sends and receives UDP messages through a socket.

Copy the programs `udpsend.c` and `udprecv.c` into your area from the `d480/dnet` directory on `turing`. You can compile them using gcc in the usual way. Also copy `scrapeskel.c`.

Please change the UDP_PORT from 1234 to another number, being 3000 + two digits from your student ID number. (If it conflicts with another student in the class, you can use a different two digit number.) Both programs should use the same port number.

The sending program prompts you for a string, then sends it via UDP. The receiving program listens on a socket for a single string, then prints it.

Testing

Verify that udpsend works by running netcat. Type `nc -u -l -p 1234` in one terminal window (except use your own port number instead of 1234) and then run udpsend in another window on the same machine.

`-u` means UDP mode

`-l` means listen

`-p` says the port number to listen

When you run udpsend you might have to type ctrl-D to terminate your message.

Verify that udprecv works by running udprecv in one terminal window and then running netcat in another: `nc -u localhost 1234` (except use your own port number)). Type one or more lines into netcat, ending with ctrl-D. Everything you type into netcat (nc) should be transmitted to the receiver and printed. You should also be able to send a file by redirecting netcat input:

```
nc -u localhost 1234 < infile.txt
```

You should be able to use `netstat -uln` to discover your udprecv receiver program listening on the designated udp port.

`-u` UDP

`-l` Programs that are listening

`-n` Show results numerically

Finally, use the two programs (run receiver first, then sender) to send a message from one to the other.

Now for the analysis.

Write these down on a separate paper. Refer to the specific examples in both programs.

1) Explain the `socket()` call: what are the meanings of its arguments in these two programs, what does it accomplish.

2) What is a `struct sockaddr_in`? What are the components of this structure that you use?

3) What is AF_INET?

4) What does htons() do?

5) What are the arguments to sendto()?

6) What is 127.0.0.1?

7) What does bind() do?

8) What does “blocking” mean? Is rcvfrom() blocking or non-blocking?

Now for some programming.

A) Change the rcvfrom() call in udprecv so that it receives the address that the message came from. Print out the sender's IP address and port number with the message.

B) Change udprecv so that if it receives a quit message it closes the socket and exits. Beware that many messages will arrive with an unwanted newline or carriage return (NL or CR) at the end that should be ignored. You need to be careful not to let that interfere with comparing the incoming message with the constant string quit.

C) Find out about the signal() C library function. (It needs the signal.h header file.) In rdprecv, use it to catch SIGINT and run your own ctrl-C signal handler function that prints "bye", closes the socket, and exits.

D) Finish the scrape() subroutine outlined in scrapeskel.c. Call it from udprecv, so that it scrapes URLs from the incoming buffers. Then print the urls that were found. Test it by sending the saved source of a web page to udprecv. There is an old source capture of the NIU web page in the d480/dnet directory on Turing.