Exercise 20

Part 1

For this program, you are going to implement an echo server in TCP and UDP, and compare their behaviors. An echo server reads or receives what the client sends, and simply echoes or sends it back to the client. This demonstrates that a connection is established properly, and that sending and receiving work, but isn't useful for anything else.

You have been given the code for an echo TCP server (both sequential and concurrent) on the class Moodle site, under Lecture 20. Modify the **`unitcp-client`** code to output only what is sent back to it by the server, then run the server and client on a port number specified as a command line argument for both programs. Following this, run **`diff`** to compare the input read in by the client, and the output produced by the client, which should, if all things work, be the same.

For instance, in one terminal window, run:

```
echotcp-server 9999
```

And in another window, run:

```
unitcp-client 127.0.0.1 9999 < <testfile.txt> > tcpout.txt
diff <testfile.txt> tcpout.txt
```

where **`<testfile.txt>`** is any file you wish to use for testing. Provided for this exercise are several files for testing. The result of the diff should be that there are no differences between the file contents sent by the client, and the file contents output by the client, after being echo'd by the server.

You can also run the server in the background, which will not require you to use separate terminal windows. To do so, enter:

```
echotcp-server 9999 &
```

(i.e., the trailing & indicates to run the job in the background). At any time, to terminate a running process which is running in the foreground, type Ctrl-C. To terminate a running background process, send it a SIGKILL signal, as follows:

```
kill -9 echotcp-server
```

Part 2

You also have two UDP servers (both sequential) given to you. Modify one of these to read input from a client and and simply echo it back, similar to the echotcp-server, but using the functions for reading and writing that are specific to UDP. Name this program echoudp-server.c.

Then in one terminal window, run:

```
echoudp-server 8888
```

And in another window, run:

```
uniudp-client 127.0.0.1 8888 < <testfile.txt> > udpout.txt
diff <testfile.txt> udpout.txt
```

Try this with at least the sample input files provided with this exercise. Are the input file and the output file always the same? Remember, UDP is not reliable delivery.

Part 3

Lastly, modify the echotcp-concurrent server so that after reading input from the client, followed by an indication there is no more input, the server "sleeps" (suspends execution) for 30 seconds (using the **`sleep(30)`** function call) before echoing the input back to the client. Then in 3 separate windows run:

```
echotcp-concurrent 7777
```

and

```
unitcp-client 127.0.0.1 7777 < <testfile.txt> > tcpout1.txt
```

and

```
unitcp-client 127.0.0.1 7777 < <testfile.txt> > tcpout2.txt
```

Or run these as 3 separate background processes.

Start the execution of the two clients closely in time, so their execution overlaps. Does the server handle concurrent requests properly?

Submitting Your Work

When you're done, submit your completed echotcp-server.c, echotcp-concurrent.c, unitcp-client.c, echoudp-server.c, and uniudp-client.c files to Exercise 20 in Moodle.