CSE 333 13su Exercise 14

out: Friday, Aug. 9, 2013
due: Wednesday, Aug. 14, 2013 by **9:00 am**.

Write the server to go along with ex13. More specifically, write a C++ program called "ex14" that accepts one command line argument: a port number to listen to.

The program should create a TCP listening socket on the supplied port number. Once a client connects, the program should read data from the connected client socket and write it to stdout until there is no more data to read from the client (i.e., until EOF or a socket error occurs). Once all of the data has been read and written, the program should close the client socket and the listening socket, and then exit.

To test your program, try running it on a specific attu machine, say attu4.cs.washington.edu, and use your ex13 binary to send it a file. For example, on attu4.cs.washington.edu, run the following command:

```
./ex14 5555 > output.bin
```

Then, on some client machine, pick a file to send to your server. For example, this command will send the 'ex13' binary to the server (assuming that the server is running on attu4):

```
./ex13 attu4.cs.washington.edu 5555 ex13
```

Once the file has transferred, you should use 'md5sum' to verify that the file was sent correctly. For example, on the server, run this:

```
md5sum output.bin
```

and on the client, run this:

```
md5sum ex13
```

and you should see the same MD5 signature printed out on both ends.

There are a few requirements on your code:

- you should modularize it nicely; consider splitting it into multiple source files if that makes sense, for example by isolating all of the network-specific code in a module.
- you should use read() to read from the input client socket. Pay attention to the possibility that read might return EINTR, and it might read less than you ask for. I recommend writing some utility functions to deal with this. Feel free to use fwrite() to write to stdout.
- write a Makefile so that we can compile your code by typing "make". Your makefile should produce a binary called "ex14".

In addition, your code must:

- compile without errors or warnings on CSE Linux machines (lab workstations, attu, or CSE home VM)
- have no crashes, memory leaks, or memory errors on CSE linux machines
- follow the style guidlines we've been using in class, including naming conventions for classes, methods, and instance variables. If in doubt, consult the Google C++ style guide (Google it to find it if you miss the link on the course web page). Cpplint is also useful for suggesting things to check.
- be pretty: the formatting, modularization, variable and function names, and so on must make us smile rather than cry.
- have a comment at the top of the source file with your name, student number, and CSE or UW email address.

You should submit your exercise using the assignment dropbox linked on the main course web page.

Computer Science & Engineering University of Washington Box 352350 Seattle, WA 98195-2350 (206) 543-1695 voice, (206) 543-2969 FAX