# ECE435: Network Engineering – Homework 1
Sockets Programming

**Due: Thursday, 4 February 2021, 9:30am**

This Homework is meant to get you started with socket programming. It should run on any Linux/UNIX/OSX machine. If you do not have access to such a system let me know and I can provide access.

1. **Download and Build the Code**

   (a) Download the code from:
   `http://web.eece.maine.edu/~vweaver/classes/ece435/ece435_hw1_code.tar.gz`

   (b) Unpack the files:
   ```
   tar -xzvf ece435_hw1_code.tar.gz
   ```

   (c) Build the C files:
   ```
   cd ece435_hw1_code
   make
   ```

2. **Test the Code**

   (a) It might be easier to see what's going on if you have two shell windows open.

   (b) In one, first run `./server`

   (c) In another, run `./client`

   (d) Type a message on client, and it should travel over the network to server and appear on the server.

   (e) Take a look at the code and see how it works.

   (f) Note, if you try to re-run the code you might find you get an error such as `Error binding! Address already in use` This is because the client ends so suddenly the network connection is not shut down properly and the network connection enters `TIME_WAIT` state which lasts 60 seconds as the OS waits in case any lingering packets are still on their way. A quick hack to avoid this might be uncommenting the `sleep()` in the `server.c` code. A more proper fix is adding a `setsockopt()` call as described in the class notes.

3. **Modify the code (7pts total)**

   (a) **Modify the server to echo** (1pt)
   Modify the server code (`server.c`) so that it gets the message from the client and sends the same message back.

   (b) **Modify the server to not exit** (1pt)
   Modify the server code (`server.c`) so that instead of exiting after one transaction, it instead loops forever reading from the file descriptor and responding

   (c) **Modify the client so that it does not exit** (1pt)
   Modify the client code (`client.c`) so it loops forever, waiting for a message to be typed then sending it. You can always use control-C to quit.

(d) **Server closes on command** (1pt)
Modify the server code so that if the string `bye` is sent, it exits the server.
You can use the `strncmp()` function for this, but beware the unusual behavior of `strncmp()` (0 means a match)
Also note that `fgets()` is going to leave the trailing linefeed at the end of the string so take that into account.

(e) **Quit client on exit** (1pt)
Once bye is echoed back from the server, detect this on the client and exit the client too.

(f) **Have the server uppercase the string** (2pts)
Modify the server so that when it receives the string it converts all of the lowercase characters to uppercase before sending back the uppercased response.
You might find the `toupper()` function useful.

(g) Some notes:

- Be sure to comment your code!
- Be sure to fix any warnings that the compiler gives.
- Be sure to handle buffer overflows. What if the client sends 100kB of raw data to your server?
- Be sure you are keeping the same connection open and *not* starting a new connection for each string. (i.e. Make sure you are looping to the right place in the server). This is tricky because in the server there are two different file descriptors involved.

4. **Something Cool (1pt)**

Do one of the following:

- Modify the server to get the port number from the command line (look into `atoi()` or `strtod()`). Modify the client to get both the hostname and port from the command line.
- When a connection comes into the server, print the port and address of the incoming client. This info can be found in the client_addr structure.
- Modify your server code to also change the color of the text that is returned. HINT: Look up "ANSI escape codes"

5. **Answer the following questions** (2pts total)

Short answers are fine. Put your answers in the `README` file using a text editor, it will be automatically included in the submission process.

(a) In the OSI reference model, which layer deals with the actual bits, voltages and frequencies involved?

(b) In the OSI reference model, which layer deals with routing packets from one network to another?

6. **Submit your work**

   - Please edit the README file to include your name.
     Also put your answers to the questions there.
   - Run `make submit` which will create a `hw1_submit.tar.gz` file containing `README`, `Makefile`, `server.c` and `client.c`.
     You can verify the contents with `tar -tzvf hw1_submit.tar.gz`
   - e-mail the `hw1_submit.tar.gz` file to me (vincent.weaver@maine.edu) by the homework deadline. Be sure to send the proper file!