Machine Problem 0
Introduction to UNIX Network Programming with TCP/IP Sockets
No due date; ungraded

The purpose of this machine problem is to familiarize you with network programming in the environment to be used in the class and to acquaint you with the procedure for handing in machine problems. The problem is intended as an introductory exercise to test your background in C programming. You will obtain, compile, run, and extend a simple network program. The extensions to the code will introduce you to one method of framing data into individual messages when using a byte stream abstraction such as TCP for communication.

We recommend that you use Ubuntu 14.04 VMs, since that is the environment the later MPs will be graded in. See the Appendix for instructions on setting them up.

Checkout your SVN directory from the class repository using (remember to replace your NETID into the path):

```
svn checkout https://subversion.ews.illinois.edu/svn/fa15-ece438/NETID/
```

You will find a folder named mp0, which contains the programs client.c, server.c, talker.c, and listener.c – all from Beej's Guide to Network Programming (http://beej.us/guide/bgnet/). Beej's guide is an excellent introduction to socket programming, and very approachable.

Compile the files using gcc to create the executable files `client`, `server`, `talker`, and `listener`. We provide a Makefile that will compile all 4 (simply run `make` inside the directory). The real assignments will require you to submit a Makefile, so if you aren't already experienced with `make`, please familiarize yourself with the provided Makefile, and figure out how you would adapt it to a new project.

Login to two different machines, and execute `client` on one and `server` on the other. This makes a TCP connection. Next, execute `talker` on one machine and `listener` on the other. This sends a UDP packet.

Note that the connection oriented pair, `server` and `client`, use a different port than the datagram oriented pair, `listener` and `talker`. Try using the same port for each pair, and run the pairs simultaneously. Do the pairs of programs interfere with each other?

Next, change server.c to accept a file name as a command line argument and to deliver the length and contents of the file to each client. Assume that the file contains no more than 100 bytes of data. Send the length of the file (an integer between 0 and 100) as an 8-bit integer . Change client.c to read first the length, then that number of bytes from the TCP socket, and then print what was received.

The client output should look like this:

```
client: connecting to hostname
client: received filelen bytes
This is a sample file that is sent over a TCP connection.
```

Where hostname is the address of the server, filelen is the number of bytes received, and the rest of the output is the file contents.

That's it. Sounds simple, doesn't it? Indeed, for experienced Unix/C programmers this MP is trivial. Others should find it a nice way to get started on network programming.

You will need to have (or quickly acquire) a good knowledge of the ANSI C programming language, including the use of pointers, structures, typedef, and header files. If you have taken CS241 you should already have the necessary background. Don't simply download the source code and compile the programs, but make sure that you read and understand how the sockets are created and the connection established. Beej's guide is a very useful tool in this sense.

This MP is optional and will not be graded, but you can use it also to test the hand in procedure using SVN. For the next assignments, we will grade only the files that are submitted through SVN. Use the names mp0client.c and mp0server.c for the modified sources that read from a file. Also within MP0 should be included a Makefile so that execution of the command `make` causes the executable code for the all the programs to be generated by the compiler. Running `make clean` instead should delete all executable files and any other temporary file that your Makefile or programs create.

To hand in your homework, first add any new file you created and you want to submit by using the command `svn add filename`.

This does **NOT** submit your homework; the next step (committing it) is what actually submits it. Once you are ready to submit your homework (i.e. have `svn add`ed all relevant files), type the following while in the directory containing the assignment:

```
svn ci -m "some resaonably informative commit message"
```

Once a file has been committed to subversion, it has been submitted. You may submit your MP in this way as many times as you'd like (with each commit replacing the previous submission). It's a great way to 'save' the work you've done so far. You can verify the files on subversion by viewing your subversion though a web browser by going to the following URL:

https://subversion.ews.illinois.edu/svn/fa15-ece438/NETID/

Appendix – VirtualBox VM Setup

The autograder runs your code in VMs – 64-bit Ubuntu 14.04 Server VMs, running on VirtualBox. Therefore, to test your code, you will need a 64-bit Ubuntu 14.04 VM of your own. (Even if you're already running Ubuntu 14.04 on your personal machine, later assignments will use multiple VMs, so you might as well start using the VM now.)

WARNING: COMPILATION CAN BE A LOT LESS PORTABLE THAN YOU THINK, ESPECIALLY WHEN OSX OR EWS IS INVOLVED. Please don't assume that it will be ok after testing it only on your personal machine or EWS. (Just don't use EWS at all; it is not well suited to classes that involve networked programming assignments.)

A tutorial for installing Ubuntu on VirtualBox can be found at http://www.psychocats.net/ubuntu/virtualbox. This tutorial is for Windows, but VirtualBox works and looks the same on all OSes. The Ubuntu 14.04 image: http://www.ubuntu.com/download/alternative-downloads. You can use either the desktop or server version – server version is a quicker download+install, and is recommended.

NOTE: during the Ubuntu install process (within the VM), there's a list of typical server programs you can choose to install; you should install the ssh server.

Use apt-get (sudo apt-get install xyz) to install any programs you'll need, like gcc, make, gdb, valgrind. I would suggest also getting iperf and tcpdump, which will be useful later.

Networking

VirtualBox's default network setup is a NAT (which we'll learn about later!) interface to the outside world, provided by the host computer. This allows the VM to access the internet, but the host computer and other VMs will not be able to talk to it. We're going to replace the NAT interface with one that allows those communications.
BEFORE YOU MAKE THIS CHANGE, though, you should use that internet access to:

```
sudo apt-get install gcc make gdb iperf tcpdump
```

Now it's time to replace the network interface. Make sure the VM is fully shut down, and go to its Settings->Network section. Switch the Adapter Type from NAT to "host-only", and click ok. Restart, and the VM will now be able to talk to other host-only VMs on the same computer, as well as with the host computer (for ssh).

Finally, `sshfs` is an excellent way to access the VM's filesystem (or any other remote filesystem). On your host system, `sshfs 192.168.56.101: directory_to_mount_in` will mount the VM's filesystem as if it were a USB flash drive. (Replacing the IP address with whatever it actually is, of course). `sshfs` may not be available on all operating systems.