**ECE/CS 438: Communication Networks** **Fall 2017**

Machine Problem 1

*Handed Out: August* $30^{th}$*,* 2017 *Due: Never*

*TA: Ashutosh Dhekne*

**Abstract**

This machine problem introduces you to socket programming in C and the mechanism for submitting assignments. We use SVN for all machine assignments. We also use virtual machines to simulate multiple network entities. This assignment will help you prepare your environment so that all subsequent assignments will be simpler to code and submit. Since this assignment has all the preparatory material, the TAs will not help you understand this part in subsequent machine problems.

# 1 Introduction

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.

# 2 What is expected in this MP?

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/fa17-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 ensure that you can adapt it to a new project. Login to two different machines (Virtual 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.

# 3 How to submit the MP?

This MP is optional and will not be graded, but you should use it to understand submission and how the autograder works. You must be able to follow this process to get your subsequent MPs graded.

To hand in your homework, first add any new file you created and want to submit by using the command `svn add filename`. Next, open `version.txt`. It should contain the number 0. Increment 0 to any higher integer. (Increment the number every time you want to request a new test from the autograder).

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

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

(What is happening here: svn keeps track of versions of files. The `add` command tells svn to start keeping track of filename, the `ci` (commit) command tells svn that it should create a new "version" of the whole repository, using the current state of all of the files it is currently tracking. You need to add a file only once. `add` alone does not put the file into the repo!) 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/fa17-ece438/NETID/` Finally, run `svn up` to see any changes (such as in the `results.txt` file) the autograder has pushed back to you. Tests generally take 1-4 minutes, and there may be a queue of students. (You can see where you are in the queue at

`https://courses.engr.illinois.edu/ece438/queue.txt`).

**Caution:** During the hours leading up to the submission deadline, the queues could be multiple hours long. So get your work done early.

# 4 How to set up VirtualBox VM environment?

The autograder runs your code in VMs—64-bit Ubuntu 16.04.3 LTS VMs (desktop version), running on VirtualBox. Therefore, to test your code, you will need a 64-bit Ubuntu 16.04.3 LTS VM of your own. (Even if you're already running Ubuntu 16.04.3 LTS 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 16.04.3 image:

`http://www.ubuntu.com/download/alternative-downloads`

After the Ubuntu install process (within the VM), you should install the ssh server. You can do `sudo apt-get install openssh-server` once the OS is installed. 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.

# 5 How to setup networking inside VMs?

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.

However, BEFORE YOU MAKE THIS CHANGE, you should use that Internet access to: `sudo apt-get install gcc g++ make gdb iperf tcpdump wget`

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.

# 6 What tips and tricks will be useful?

Copy-pasting directly from pdf files is a bad idea. Dashes, quotes, and kerned characters may get completely mis-represented when pasted in a terminal.

Never add compiled executables and object files to the svn. You might lose points in subsequent assignments if you add executables or object files to svn. Use `svn add` to add only the source files (`src` folder), `Makefile`, `version.txt`, and `teamname.txt` files.

MP0 is ungraded, but still very important to get started. Performing this assignment successfully will make submitting the subsequent assignments much easier.