**ECE 4380/6380: Computer Communications** **Spring 2018**
**Machine Problem 1** **Due: 11:59 pm, Thursday, February 8**

## Introduction to Unix network programming with sockets

The goal of this machine problem is to familiarize you with the basics of network programming. You will obtain, compile, run, and extend a simple network program. You need access to two unix machines so that you can verify that you are connecting over the network.

An easy way to have access to multiple machines is to use the Clemson University College of Engineering and Science workstations. If you don't have an account already check with the Unix help desk about obtaining one. To request an account send email to coes-unixadm@clemson.edu (see also http://cecas.clemson.edu/help/)

(a) Make a directory in your home directory and download the programs `client.c`, `server.c`, `talker.c`, and `listener.c` from Beej's Guide to Network Programming, available at

http://beej.us/guide/bgnet/

Read Beej's guide (only Sections 1 through 6) to determine how the programs work. Note that the C files are linked inside the HTML guide. Compile the files to create the executable files `client`, `server`, `talker`, and `listener`. For example, to create `client` do

```
gcc -o client client.c
```

if you are using Linux.

Login to two different machines and execute `client` on one and `server` on the other. Do the same with `talker` and `listener`. For example, if you are logged into apollo05, open a new terminal window and execute "`ssh apollo04`". In the apollo04 window, verify that no one else from out class is using the machine by executing "`who`". If the machine is free, execute "`./server`". Then execute the client program on apollo05 making a connection to apollo04.
Notice that the connection oriented pair, `server` and `client`, use a different port than the datagram oriented pair, `listener` and `talker`. Modify the code so that both pairs of programs use the same port, and simultaneously run `server` and `listener` on one host, and `client` and `talker` on another.

(Check if another student in our class is logged into either of the machines with `w` or `who` and if so coordinate a time so you are both not running the programs at the same time. If another student is running `server` and you start `server` you will get an error message from the `bind` command: `bind: Address already in use`. This is because you are both running an application that tries to use the same port to open a TCP socket. One solution is to pick a different port number. You will get a different behavior when trying `server` and `listener`.)

**Question 1: Do the pairs of programs interfere with each other? Explain why or why not.**

(b) This example of a server is so simple that we can also use a web browser to get the "hello, world!" message. Continuing the example from part (a), run `server` on apollo04 (or the machine of your choice) and open a browser window on a different machine (even with Microsoft Windows!). In the browser window type

http://apollo04.ces.clemson.edu:3490/index.html

(Notice by adding the ":3490" to the address, the browser opens a TCP socket to port 3490 instead of the default port of 80. Obviously, the number you enter in the browser should match the port number used by `server`.)
When your browser opens a TCP connection to our simple server, it sends a HTTP command to the server. Modify the `server.c` program to print out the commands that the browser sends to the server. Note that

you will need to add a `recv` command to the `server.c` program to get the message that the browser sends. See the `client.c` file for an example of how to use the `recv` command. Read section 9.1.2 in Peterson and Davie (pages 650-657) for an overview of the HTTP commands.

If you are using the Apollo or unixlab machines the firewall may block packets from a client that is not on the same subnet as the server. Either launch firefox on the same machine you use for the client, or use the command line program `wget` (e.g., `wget http://apollo04.ces.clemson.edu:3490/index.html`). The `wget` program will issue the same get command as a web browser does but stores the information returned by the server in a file instead of displaying it in a window. `Wget` will report an error because our server program is not following all of the details of the HTTP protocol, but the information the server receives from `wget` will be the same as the information the server receives from any browser.

Many new browsers no longer display the "hello, world!" message because our server is not sending correctly formatted HTTP commands. If your browser does not display the text, then use `wget`. However, for this question, we do not need to see what your browser or `wget` receives. Instead, we want to see what the server receives. The information received will not depend on if you use a browser or `wget`.

**Question 2: Print a copy of the information the server receives. Identify the START_LINE and MESSAGE_HEADER that your new server receives from the browser. See the description of the request and response messages in section 9.1.2, and identify these fields in your print out.**

(c) Modify the `client.c` and `server.c` programs so that the client program sends an identification number to the server, the server looks up the identification number in a table, and the server returns a string associated with the identification number back to the client. The client should print the identification number and the string that is received from the server. Both the client and server should verify that the identification number contains exactly 6 digits, and the client should read the number from the command line similar to the method used by `talker.c`. The client should not initiate a request to the server if the identification number is invalid (e.g., the wrong number of digits or characters other than 0-9). The server should return a message indicating that the number is invalid if it is not found in the server's table. For simplicity just generate a table with 5 identification numbers and return strings pairs for the server to use. Note we will test your server with our client, so your server will receive invalid numbers.

**Hand In for part (c):**

Use the names mp1client.c and mp1server.c for your version of the programs that exchange identification numbers and strings as described in part (c). A makefile has also been provided and your code must compile when this makefile is used. Put a copy of the makefile in the directory with your code. Running the command `make` causes the system to read the makefile and compile your source code to create the executable code. Also, create a file containing the answers to questions 1 and 2 above and in addition the following:

This part should briefly describe the implementation history for your client and server programs for part (c). In particular, you should briefly discuss (1) how does the server protect itself from a client that sends invalid information; (2) what is the limit on the size of the largest string that the server can send; (3) how does your code ensure that the client has sufficient memory to receive the string; (4) how does the client protect its self if the server incorrectly sends a string that is too big. In addition, you should acknowledge the sources if you copied sections of your code from outside sources (even if you modified it later). You should credit Brian "Beej" Hall.

When you get the C programs and answer files completed, you are to submit the files to the assignment area of Canvas. Note you do not need to submit the C program you use to answer question 2 above. Only the C programs for part (c) need to be submitted.

While you are encouraged to discuss the general approach of the problem with others, you are expected to do your own coding and write your own lab report. You will hand in the code you have modified for grading, and the code may be scanned with the MOSS tool http://theory.stanford.edu/~aiken/moss/ . This tool compares each implementation and identifies those that are identical or nearly so. This may lead to the conclusion that you copied your code from someone else or that someone copied your code. In either event, you will receive a zero for the assignment. If additional instances of cheating are discovered you will fail the class and further action may be taken by the University. It is your responsibility to protect your code.

The machine problem is worth 100 points. For each day your submission is late your score will be reduced by $2^i$ points where i = 1 for a submission after the due date but before 11:59 pm on the next day (i.e., within the first 24 hours). Submission after 24 but before 48 hours sets i = 2 for a 4 point late penalty, etc. (Note that this rule applies to machine problems only. Late homework sets are not accepted.)

## Tips for those new to Linux or could use a refresher

Clemson has a site license to "SSH Secure Shell", a Windows program to connect to remote Linux machines. From the Clemson CCIT web site look for software downloads.

There are many online tutorials on how to use the command line. Here is one good one:
http://linuxcommand.org/learning_the_shell.php

An alternative approach is to install a virtual machine on your own computer. While this takes considerably more time to setup, having experience managing your own Linux machine will be valuable in future courses. Install either VitrualBox, or VMWare Workstation Player. Then use one of these programs to install the latest version of Ubuntu as a virtual machine.