Machine Problem 0

Introduction to UNIX Network Programming with TCP/IP Sockets

Due: no due date, this MP will not be graded

Please read all sections of this document before you begin coding.

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 on the EWS workstations, then hand in some files. 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.

Specification

We recommend that you use an EWS Linux machine since our auto-grader program will check your MP submissions on these machines. Information about the workstations is available on the homepage:

http://ews.uiuc.edu/

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/sp14-cs438/NETID
```

(if you are not familiar with SVN please reference `http://svnbook.red-bean.com/`). You will find a folder named `MP0`, which contains the programs `client.c`, `server.c`, `talker.c`, and `listener.c` from Beej's Guide to Network Programming, also available at:

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

Figure out what these programs are supposed to accomplish. Reading Beej's guide itself is of course very helpful, if you can tolerate his sense of humor. Compile the files using the Gnu C compiler to create the executable files `client`, `server`, `talker`, and `listener`. For example, to create the executable file `client` you'd execute:

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

Learn how to use the `make` command (on a file you create, the default file name is `Makefile`) in order to simplify the task of compiling files. Login to two different machines, and execute `client` on one and `server` on the other. This makes a TCP connection. 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 simultaneously run server and listener on one host, and client and talker on another. 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 (assume that it is a text file, and ignore additional data). Send the length of the file (an integer between 0 and 100) as a 32-bit integer[1], followed by the data bytes. Change the client to read first the length and then the amount of data specified by the length from its TCP socket. You may want to use the `read` and `write` system calls in place of Beej's calls to `send` and `recv`.

[1] In network byte order, if you already know what that means. Ignore this comment if you do not.

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 content.

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.

Discussion

Of course the EWS workstations are already well connected by the network file system, a client-server network application. Thus you see almost the same set of files no matter which machine you are logged onto. You can communicate from one machine to another by writing a file with one machine and reading it on another. For example, you can compile all the files on one machine, and then run the resulting executable files from any machine. When you are writing network software for this class, you need to pretend the machines are separated, so you feel like you are accomplishing something.

Hints

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. Get a book on the subject if necessary (the class web page has suggestions). 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. The Beej's guide is a very useful tool in this sense.

Open several windows on one machine and use `ssh`, `rlogin`, or `telnet` to remotely run commands such as `client` and `server` on other machines. If you are using the EWS machines, for example, you can ssh to `linux1.ews.illinois.edu` and `linux2.ews.illinois.edu` and run the server on one machine and the client on the other.

Help with C and Unix system calls and commands is provided by the online manuals. For example, executing the statement `man bind` tells you about the system call `bind`, which is included in the C programs you are to download for this machine problem. The man pages for a system call tell you about the header files you need to have included in your program to use the system call, and they also specify libraries to link in when the program is compiled.

Hand In

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. The executable files should have the same name as the source file they are compiled from, and should be in the same `MP0` folder. 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, you need to follow the next step to commit it.

Once you are ready to submit your homework, type the following while in the directory containing the assignment:

```
svn ci –m “”
```

You may commit your MP as many times as you'd like. It's a great way to 'save' the work you've done so far. We will use the last submission you made for the MP when we grade your MP. Once a file has been committed to subversion, it has been submitted. You can verify the files on subversion by viewing your subversion though a browser by going to the following URL:

https://subversion.ews.illinois.edu/svn/sp14-cs438/NETID/

A file cannot be graded if it has not been committed. Failure to commit a file on time will result in your MP being considered late. It is your responsibility to ensure all your work is committed by the due date.