CS/ECE 438: Communication Networks for Computers Spring 2010
Machine Problem 1 Due: 9:00 PM, Friday, February 5th

Introduction to UNIX Network Programming with TCP/IP Sockets

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 CSIL 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.

Of course the CSIL workstations are already 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.

Problem 1: Initial setup

You are to use the UIUC College of Engineering CSIL workstations. Information about the workstations is available on the homepage

```
http://csil.cs.illinois.edu
```

Make a directory called MP1 somewhere inside your main directory and download the programs `client.c`, `server.c`, `talker.c`, and `listener.c` into it from Beej's Guide to Network Programming, available at:

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

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?

Problem 2: Some simple extensions

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 (ignore additional data or issue an error message). Send the length of the file 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.

Problem 3: Building a proxy

In computer networks, we often use *proxies* to act as intermediaries on connections. For example, *web proxies* are used to deny access to URLs that are part of a blacklist, and maintain caches to service requests locally to reduce load on origin servers. There are also anonymizing proxy servers (which hide identity of clients from servers), gateways (which translate between different networks running different protocols), as well as proxies that filter content and monitor traffic for accounting purposes. In this problem, we'll implement a very simple proxy that sits between the streaming server and client code (so connections from the server are "tunneled" through the proxy before reaching the client. To keep things simple, we'll just do this for the streaming client/server, not for the datagram client/server.

Create a new program called `proxy.c`, which accepts a command line argument specifying the IP address and port of a remote host. This program is to act as a proxy between the server and the client – that is, the proxy will receive the file contents from the server, and forward them on to the client. Do not make any changes to the client or server code to do this, as any such changes are unnecessary (the server should not be able to distinguish the proxy from a regular client, and the client should not be able to distinguish the proxy from a regular server. Your proxy should immediately forward all data it receives from the server to the client. For example, you shouldn't read the entire file from the server, store it locally, and then send the result to the client – upon receiving data from the server, the proxy should immediately forward that data so as to reduce delay. When you're done, run your code on three different machines: the client on one, the proxy on another, and the server on a third. Send a file from the server to the client through your proxy to test it out.

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. Get a book on the subject if necessary (the class web page has suggestions).

Early on, make sure that you can comfortably edit files on the workstations. Depending on how you access the workstations (you can do so remotely), you may need some adjustments in your `.login` and `.cshrc` files. You can see/set various settings with the `set` and `setenv` commands. You may want to select a different shell program (which controls interactions between you and the workstation) than the default shell. The turbo C shell tcsh is popular these days.

Open several windows on one machine and use `ssh`, `rlogin`, or `telnet` to remotely run commands such as `client` and `server` on other machines.

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

Use the names `mp1client.c, mp1server.c and mp1proxy.c` for the version of the programs that reads from a file. These files should be placed in your directory MP1. Also within MP1 should be included a makefile so that execution of the command `make` causes the executable code for the programs to be generated by the compiler.

Finally, create a file README containing two parts.

- *Part 1: Implementation Log*
  The part should briefly describe the implementation history for your client and server programs. In particular, you should briefly discuss the main design decisions you make in implementing the computer code; for example, how do you handle overly long files? In addition, you should acknowledge the sources if you copied sections of your code from outside courses (even if you modified it later). *For this machine problem, your log should credit Beej, describe (i) your change to the port numbers (ii) your extension to send a file across the TCP connection, and (iii) your extensions to build the proxy.*

- *Part 2: Experimental Results*
  Explain the outcome of your experiments. What happened when you ran `server` and `listener` on the same host listening to the same port, then ran `client` and `talker` on another host? What happened when you ran your proxy, was the time required to send the file significantly increased?

From within the directory, type

```
~ece438/Handin/handin
```

Follow the prompts and hand in the C programs, the README file and makefile. (Please check the course newsgroup for tips in case our handin program isn't running smoothly.)

Grading Scheme

10 pts Code

- Well commented
- Meaningful variable & function names
- Readable and indented
- Acknowledge sources

20 pts README

- Implementation Log
- Acknowledge sources
- Experimental Results
- Plain text file

10 pts Working Makefile

- Generates all executables with single execution of "make"
- No errors

30 pts working mp1server

- Handles multiple connections
- Checks command-line arguments

30 pts working mp1client

- Checks command-line arguments
- Correct output