Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.033 Lab, Spring 1999 Handout 3 — A TCP Proxy Server February 12

Due date: February 22

1 Introduction

The first programming project in 6.033 is meant to prepare you for the next two projects by acquainting you with UNIX programming, especially network programming. Unlike the other labs, we will start you off with some code for setting up network connections, since the code for doing this is very mechanical. From this point on, you will build a fully functional TCP proxy.

2 Running a simple client/server program

We will use a trivial client/server program as our stepping stone. The C sources for the program can be found in the 6.033 locker in the `lab/src/one/cli-serv` directory. You should copy the contents of the directory to a directory under your control.

Now, build the client and server. Start the server by typing `./server`. In another window, type `./client` *name*, where name is the name of the machine on which the server is running. You can find out the name of a machine you are using into by typing `hostname`. Every time you run the client, the server will print the notorious "hello, world!" message.

3 Understanding the client and the server

Once you get the programs running, you should study the code and try to understand it. You might find it helpful to read chapter 10 ("Communication in distributed systems") of the Tanenbaum book to obtain a general understanding of client/server programming. The specifics of each UNIX networking call can be found in the manual pages (e.g., type `man socket`). If you cannot make sense out of the program using the sources, Tanenbaum, or the man pages, you might want to borrow "UNIX Network Programming," written by Richard Stevens; or, get in touch with your TA.

4 TCP Proxy Server

A TCP proxy server is a server which acts as an intermediary between a client and another server, called the destination server. Clients establish connections to the TCP proxy server, which then establishes a connection to the destination server. The proxy server sends data received from the client to the destination server and forwards data received from the destination server to the client. Interestingly, the TCP proxy server is actually both a server and a client. It is a server to its client and a client to its destination server.

A TCP proxy server can be useful to get around services which restrict connections based on the network addresses. For example, there are many servers at MIT which will only serve data to addresses within the MIT network. By running a proxy server on the MIT network, clients from outside MIT can use those servers by connecting through the proxy server. The MIT servers will think they are serving data to a machine on the MIT network (namely the proxy server machine) and they will be right. However, the proxy forwards the data out of the MIT subnet, thus subverting the protection mechanism. This is a violation of the rules of use on most restricted servers, so we do not encourage you to do this.

The proxy server you will build for this lab will be invoked at the command line as follows:

`% ./proxy` *destination-host destination-port listen-port*

For example, to redirect all connections to port 3000 on your local machine to the MIT web server, do:

```
% ./proxy web.mit.edu 80 3000 &
```

The proxy server will accept connections from multiple clients and forward them using multiple connections to the server. No client or server should be able to hang the proxy server by refusing to read or write data on its connection. For instance, if one client suddenly stops reading from the socket to the proxy, other clients should not notice interruptions of service through the proxy. You will need asynchronous behavior, described in the "Using TCP Through Sockets."

4.1 Requirements

Before you implement the proxy server, we require you to implement a library for asynchronous I/O. Copy the framework from `lab/src/one/async` and `lab/src/one/multifinger`. We have provided an interface; you just need to implement four functions in `async.c`. The "Tools of the Trade" handout describes how to make a library. To manage callbacks, the `libasync.a` library must implement the four functions below:

- `void cb_add (int fd, int write, void (*fn)(void *), void *arg);`

  Requires: `fd` is a valid file descriptor
  Effects: If `write` is 0, tells the dispatcher to call function `fn` with argument `arg` when file descriptor `fd` is ready for reading. Otherwise, tells the dispatcher to call function `fn` with argument `arg` when file descriptor `fd` is ready for writing.
  Modifies: `rfds`, `wfds`, `rcb`, `wcb`

- `void cb_free (int fd, int write);`

  Requires: `fd` is a valid file descriptor
  Effects: Tells the dispatcher it should no longer call any function when file descriptor `fd` is ready for reading (if `write` == 0) or writing (if `write` != 0).
  Modifies: `rfds`, `wfds`

- `void cb_check (void);`

  Effects: Wait until one or more of the registered file descriptors is ready, and make any appropriate callbacks. A callback executes a function with arguments set by `cb_add`. When `cb_check` detects several ready file descriptors, it must make a callback for each ready descriptor before `cb_add` returns.

  Note, this is very tricky compared to the other three functions. For instance, a callback could recursively call a function in this library.

- `void make_async (int s);`

  Requires: `fd` is a valid file descriptor
  Effects: Make the socket `s` non-blocking.

We highly recommend that you read "Using TCP Through Sockets" before attempting to code the library. After your library works (try linking it with the `multifinger` program), design and implement the TCP proxy as given above.

5 How/What to hand in

When you are ready to hand in the lab, copy your project to the `/mit/6.033/lab/handins/p1/`*username* directory, where *username* is your Athena user name.

The project should include a valid Makefile and the source file or files for both the TCP proxy and asynchronous library. The Makefile should create a proxy server named `proxy` and a library named `libasync.a`. We will test both your TCP proxy and asynchronous library. If your library fails to work properly, we will try your TCP proxy with our own asynchronous library. You can place comments or directions in a README file.

We will post further details to 6.033-lab@mit.edu. Moreover, we encourage you to discuss issues on the zephyr instance and 6.033-lab@mit.edu. Closer to the due date, we will release a program to test the robustness of your TCP proxy.

6 References

Stevens, W. Richard. *Advanced Programming in the UNIX Environment.* Addison-Wesley, 1992.

Stevens, W. Richard. *UNIX Network Programming.* Addison-Wesley, 1990.