Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.033 Lab, Spring 1995-96 — Handout 2 — UNIX programming — February 6

Due date: February 16

1 Introduction

In preparation for the two programming projects you might want to get some experience with UNIX network programming, since both projects use the UNIX networking support to send and receive messages. This handout contains a trivial exercise to walk you through the steps involved in sending and receiving network packets under UNIX. This exercise is quite detailed; subsequent handouts will assume this basic knowledge and will not be as detailed.

We expect you to do this exercise on your own before the first recitation section about the lab on Friday February 16. You do not have to hand in anything and you will not receive a grade; however, the projects build on this exercise, so do not skip it. This little exercise should take you at most a couple of hours.

2 Running a simple client/server program

You will be modifying a trivial client/server program. The C sources for the program can be found in the 6033 locker in the subdirectory lab/src/un.

The locker contains 3 files: client.c, server.c, and Makefile. Copy the files to you own locker and type "make." Make will compile client.c, server.c, and create the binaries client and server (type "add cygnus" to add gcc to your PATH). Start the server by just typing "server." In another window, type "client *name*,", where name is the name of the machine on which the server is running (i.e., in this the case the machine on which you created the window with the server. (You can find out the name of a machine on which you are logged into by typing "hostname".) Every time you run the client, the server will print the notorious C welcome message.

3 Understanding the client and the server

Once you get the programs running, you should study the code and try to comprehend what is going on. You might find it helpful to read chapter 10 ("Communication in distributed systems") of the Tanenbaum book—one of the 6.033 text books—to obtain a general understanding of client/server programming. The specifics of each UNIX networking call can be looked up in the man pages (e.g., type "man socket"). If you cannot make sense out of the program using the sources, Tanenbaum, and the man pages, you might want to borrow "UNIX network programming," written by Stevens; or, get in touch with your TA, Dawson Engler.

4 Modifying the client and the server

The programs client and server give you a skeleton to write any client/server program. As the last part of this exercise, write a more interesting client and server. For example, you can turn the server stepwise into a compute server:

1. Modify the client to read an expression from standard input and then send this expression to the server.
2. Modify the server to compute and print the result of evaluating the expression that was sent by the client.
3. Modify the server to send the result of evaluating expression to the client, and modify the client to print the reply.

The result of these simple changes will be a client that acts as the front end, reading and printing expression, and compute server that evaluates the expression read by the client.

There are a couple of pitfalls to watch out for:

1. Reading input from the screen and tty requires some UNIX specific knowledge. The function *scanf* can be used (type "man scanf").
2. The *read(s, msg, MAXMSG)* call in server.c reads blocks until it has read *MAXMSG* bytes or until the connection is broken by the client (e.g., by calling *close*). So, if you do not want to close the connection and you want to send less then *MAXMSG* bytes, you have to work out some protocol between the client and server (e.g., send first how many bytes you are going to send, and then the data itself).
3. Machines have different byte orders (see Tanenbaum Chapter 10 and "on holy wars and a plea for peace", by Cohen, reading 19 from the reading list). So, if you send an integer from one machine to another, the value at the destination may be different than at the source. To avoid this situation, you have to encode data objects larger than a byte into a standard byte order (e.g., ints, longs, shorts). UNIX has library routines for doing so: *ntohs* etc. ("ntoh" stands for "network to host order", UNIX also provides a set of complementary functions to convert from host order to network order "hton"; type type "man ntohs" for more information.). Test that you are performing this conversion correctly by sending integers between machines with different byte orders (you should write a small program to test whether the machine you are running on is little endian or big endian).