Lab 08: Socket Programming

OSL, Mon, Oct 15, 2012

Objective:

1. Get familiarized with network programming.

General instructions:

1. This lab is to be done in **groups of two students.**
2. You are **not allowed to exchange code snippets or anything** across groups. In case of any doubts, you are allowed to consult any Internet resource, books, or the course TAs. While it is alright to read on general socket programming on the Internet, and look at code examples, this should be only for the purpose of understanding. Do all the coding yourself, do not copy or cut-paste code from any website.
3. You can use C/C++ for coding, **no other language**. Although Java/Perl and other languages may provide a better interface, C/C++ are bare-bones, and hence appropriate for a networking course.
4. Pay attention to the variable names, function names, file names, directory names, etc. Make sure they are intuitive. Ensure that your code is documented and easy to follow.
5. We will evaluate all of the above aspects of your code, not just whether or not the code runs.

Lab Instructions

1. You have to implement a simple client-server application. The client sends a sentence to the server. The server converts the sentence into capitals and returns it to the client. For example if the client sends “This is a Lab”, the server should return “THIS IS A LAB”. See slide 26 of the lecture slides http://www.cse.iitb.ac.in/~sri/cs348/cs348-lec26-28-Sockets-2012.pdf
2. The client takes two arguments: the name of the server machine and the port number to which it should connect (in that order). It first translates from the servers name to its IP address, using the *gethostbyname* system call. It then opens a socket to the server using the *socket* and *connect* system calls. Next, it reads the string from the user input, sends the string, using the *write* system call. Then, it reads the reply, using the *read* call and displays the result to the user, and loops for the next string. It closes the connection after 3 iterations.
3. The server takes one argument, which is the port number on which it listens for an incoming command. It uses the *socket, bind,* and *listen* system calls to create a socket, bind the port number to it, and to listen for a request. On getting the request, it *accept*s it, *read*s the message sent to it, executes the conversion and uses the *write* call to send the results back to the client.
4. **Assumptions:** Since the goal of this lab is get familiar with socket programming, you can assume that the input string will be at most 20 characters long and that too consisting of only alphabets and spaces. If you make any other assumptions, state them in the README.txt to be submitted.

Submission guidelines

Create a directory <rollnumber1>_<rollnumber2>_lab08. This should contain the following files:

1. A file called "README.txt" which gives
    1. List of relevant files: including all source files and configuration files which *you* have written. Of course you need not include things like stdio.h which someone else has provided for you. Specifically *do not* have any irrelevant, old, or temporary files which clutter the directory.
    2. Compilation instructions: How should one go about generating the executable from your source files? Give the actual set of commands which someone can cut-paste from the README in order to compile. Provide such instructions for each executable file you have to generate.
    3. Running instructions: You may be generating more than one executable. Describe logically what each executable does. In addition, for each such executable file, how should one run it? What are the command line arguments (describe each argument)?
    4. Test cases: Give three test inputs for which your application works correctly.

2. The various files listed in 1.1 above.

Now tar it as follows:
tar -zcvf <rollnumber1>_<rollnumber2>_lab08.tgz <rollnumber1>_<rollnumber2>_lab08/

Submit the file <rollnumber1>_<rollnumber2>_lab06.gz via moodle for grading.

We willl run your client-server application to verify that it works on your test cases and then test it on some cases of our own. We will also evaluate the cleanliness of your coding, documentation, and commenting.