CS 360 Special Topics: Computer Networking

Spring 2007 **DUE: Feb 13, 2007**
Assignment #1 **20 points**

Introduction to Socket Programming

The focus of this assignment is to become familiar with the Berkeley UNIX socket networking interface. In particular, this assignment requires the use of Internet-domain sockets and UDP for communication between a client and server. For this assignment, you will need to use the machines in the CS lab booted into Linux or zeus/ada for little-endian machines and circe for a big-endian machine. Remember, you are required to use Subversion to manage your source code.

The Network Calculator

You are to write a client and a server for this assignment. The server will accept requests to perform simple math operations on integer values and return the result. The client will parse a request to perform a math operation from its command line arguments and send this request to the server. The client will display to the screen the result sent back by the server.

The server will always return a 32 bit integer result for add, subtract and multiply requests. For division, the server will return either a 32 bit floating point number or a 32 bit integer, whichever is appropriate. For example, 100 divided by 10 should return 10, but 5 divided by 2 should return 2.5. On zeus, both an `int` and `float` are 32 bits.

The MathPacket

To perform this operation, we need to define a *protocol* for requests and responses and define how the data will be transferred. The client and server will each send and receive a *MathPacket*. The layout of the MathPacket is as follows, each of the following dashes represents one bit:

```
|<-------------32 bits ----------->|
|--------|--------|----------------|
|PackType|Version |xxxxxxxxxxxxxxxx|
|               Operand One         |
|               Operand Two         |
|-----------------------------------|
```

The bits marked with an ***x*** are unused and may be used in a future version of the packet.

`Version` should be set to: `00000001` for this assignment.

`PackType` is as follows:

```
00000001        Request Add
00000010        Request Subtract
00000100        Request Multiply
00001000        Request Divide
11111110        Result of an Add
11111101        Result of a Subtract
11111011        Result of a Multiply
11110111        Result of a Divide (integer result)
11110110        Result of a Divide (float result)
01111111        Error: invalid PackType
10111111        Error: invalid Version
11011111        Error: invalid packet size
```

`Operand One` and `Operand Two` are both 32 bit fields. Any type of `Request` packets will provide an integer in each of the `Operand One` and `Operand Two` fields. The `Result` packets will return the result of the operation in the `Operand One` field. The value in the `Operand Two` field in `Result` packets is unspecified and should be ignored by the client.

The fields in the MathPacket should be transmitted in **big-endian** order. Intel CPUs are little-endian. You should use the `hton`*X* and `ntoh`*X* family of functions to achieve this.

There are a number of ways to correctly implement the `int/float` dichotomy for `Operand One`. The solution is left to the implementer.

The Client

The client should take 5 command line arguments: the IP address of the server, the port the server is listening on, the operation to perform (a, s, d, m), and two integers to serve as `Operand One` and `Operand Two`, in that order. The client should send one request, print the result and then terminate. The client should print the following message upon success, where the # is replaced with the value returned from the server:

```
Result: #
```

Upon receiving an Error packet back from the server, the client should display one of the following error messages:

```
Error: Wrong Packet Version returned from server
Error: Invalid Packet Type
Error: Invalid Packet Size
```

The Server

The server should take one command line argument: the port on which to listen. The server should loop forever handling requests. The server should return a packet of type `Error` if the request packet has an invalid `PackType`, invalid `Version`, or is the wrong size. The server should not print anything to the screen.

Reference Implementation

The goal of establishing a common protocol and packet layout is to allow anyone's client and server to interact with anyone else's. To facilitate this, you may send requests to: zeus (64.59.233.197) port: 9998 to test against the sample solution server. Zeus is a little endian machine. Send requests to Circe (64.59.233.204) port 9998, to test against a server on a big endian machine.

What and How to Submit

You must submit TWO .c files, one for the client and one for the server, and ONE .h file that contains the data structure for the MathPacket and any extra data or `#defines` you deem necessary. The source code should be well documented and conform to the Coding Standards. You will also need to submit ONE Makefile that will build your client as an executable file named `client` and build your server as an executable file named `server`. The names of the targets for these two files should be client and server, respectively. The Makefile should also build both client and server by default if no command line arguments are given to the `make` command. You also need to submit the **.svn** directory that contains your subversion information.

You will need to roll all FOUR of these files and ONE directory into a gzipped tar file named **PUNetID_cs360s07_PA1.tar.gz** and submit that electronically using the submit script on **zeus** as detailed in lecture. You also need to turn in a (color) hard copy of all FOUR files at the beginning of class on Feb 13th. Both the electronic and paper submissions must be made by 1pm, Feb 13th.

How will this be graded?

Your client and server should work for all permutations of
- the client is on a little- or big-endian machine
- the server is on a little- or big-endian machine
- the client and server are both running on the same machine
- the client and server are running on different machines.

Incorrect format for the output of the client: lose 5 points
Does not work with the reference implementation: lose 10 points

Hints

You should define the data structure (a **`struct`**) for the MathPacket in a header file that can be shared between the client and server. This header file should also `#define` some things to make your code **readable**.

You will need to look at the `man` pages for `socket, inet_addr, htons, recvfrom, sendto, close`, etc.

You should be able to run the client and server on the same machine or on two different machines and achieve the same result. Remember, the IP address for the **localhost** is 127.0.0.1.

In order for multiple people to work on this assignment at the same time on the same machine, each of you will need to stay to your assigned ports (http://zeus.cs.pacificu.edu/chadd/cs360s07/ports.html).

When you `recvfrom` a socket, the 5th parameter gives you the address and port the data was sent from. This data structure may be useful to send a response back to the machine that just sent you data.

To test your code on a big-endian machine, use circe.cs.pacificu.edu (64.59.233.204). Be sure to test a client running on a little endian machine talking to a server on a big endian machine, and vice-versa.

Extra Credit

+2 points: have your client accept a DNS name or IP address as the first command line argument.

Start on this today! Come see me with questions tomorrow and Monday!
Be prepared to demo your client and server in class on Feb 13th!