

Lab 06

Web Application and Server Programming

Goals

This “lab” is a bit different from previous labs in this class, and is more in the nature of a programming assignment. There are 2 parts to the programming assignment. Extra credit is available for both parts 1 and 2, and will be worth an extra 5 points in each case.

Instructions

For this and all future labs, all of your work will be done and submitted through your own student repository on the nisl class server. If you have not done so already, you need to check out a working copy of your personal repository before you can begin this lab.

Exercise 6.0: Preliminaries

Check out a working copy of your students repository if you haven't already done so. Your student repository will have a url designation of the form:

<http://nisl.tamu-commerce.edu/repo/csci553/username>

Where you need to substitute your own nisl account username for the last portion above.

In your working copy, create and add to your repository a directory called *lab06* (make sure you name it exactly as shown, make sure you not only create the directory, but you run the appropriate command to add the directory to be under version control). All work for this lab is to be added and checked into the *lab06* directory of your repository.

Exercise 6.1: Simple RPC-like Server Implementation Using CGI

XML-RPC is a remote procedure call technology that allows you to perform distributed computing by invoking procedures that actually get executed on a remote computer while the answers are then returned to the original caller. XML-RPC actually works by marshalling procedure calls into XML descriptions, sending the description over the HTTP protocol to the RPC server, which decodes the procedure description, executes the indicated procedure, then marshals the result again as XML formatted data to return back to the caller.

In this first task, you will need to implement a small CGI script that can accept commands to do simple addition, and display the results on a page. We will use URL field values to pass the procedure and parameters to be operated on to your CGI script.

For example, if you want your CGI script to add together the value 42.5 and 23.1 you might construct a URL that looks like:

<http://nisl.tamu-commerce.edu/cgi-bin/harry/arithmetic.py?operation=add;op1=42.5;op2=23.1>

Your page/cgi script should calculate and then simply display the answer such as:

Answer: 65.600000

Your CGI script should support the following operations: add, sub, mul, div

You will need to use the `cgi.FieldStorage` class (if you are implementing in Python) to easily read the operation and `op1/op2` field values passed in by the URL. You might also want to use the `cgilib` module to enable viewing of traceback/exceptions generated by your script while you are creating and debugging it.

You will need to develop and debug your script in your student `cgi-bin` directory. Please name your cgi script `arithmetic.py`. Once you have your `arithmetic.py` script working, you should put it into your `lab06` directory and check it in.

Extra Credit: I will give extra credit for handling the following problems. Handle division by 0 errors for the division operation. Also handle badly formatted number numbers (e.g. `op1=x`). One easy way to do this, especially the second error

case, it to catch the appropriate exceptions if they occur. In both cases you should display a string indicating the error that occurred, for example:

Answer: Badly Formatted op1 (x)

Exercise 6.2: Simple RPC-like Client for Calling your CGI script

In part 1 of the exercise, you built a simple CGI server script that can perform basic arithmetic operations. The second half of performing a XML-RPC remote procedure call is when the server marshals the result/answer back into xml and returns it back to the client to unmarshal and determine the result of the remote procedure call. We will do something similar by parsing/scraping the resulting html from invoking your CGI script to determine the answer.

A strong hint for this portion of the exercise. In lecture 16 we showed an example of a small python script (called httpex.py) that would open a socket connection to an http server, and display the raw http return header that it received back from the server. You can perform a large chunk of part 2 of the exercise by reusing this code. For the client portion, you need to contact your CGI script on the nisl server with an appropriately formatted url string to perform some arithmetic operation. You then need to do some simple parsing of the data lines returned by reading the socket to find the calculated answer.

You should call your client arithmeticclient.py and implement the following 4 functions:

```
def add(op1, op2):
    @param op1 and op2 are floating point numbers
    @returns a float value, the result of adding op1+op2
def sub(op1, op2):
    @param op1 and op2 are floating point numbers
    @returns a float value, the result of subtracting op1-op2
def mul(op1, op2):
    @param op1 and op2 are floating point numbers
    @returns a float value, the result of multiplying op1*op2
def div(op1, op2):
    @param op1 and op2 are floating point numbers
    @returns a float value, the result of dividing op1/op2
```

The result of each function should be to format a http get header with appropriate URL parameters to perform the arithmetic, open a socket and connect to the nisl HTTP server, then parse the returned lines read from the socket for the answer (finally returning the answer that was found). Another strong hint, the parsing of the returned HTTP header is the same in all 4 cases, you are simply looking for a line that starts with "Answer:" then what follows needs to be converted into a floating point value and returned as a result from the function. The only difference in each of the 4 functions is that the HTTP get header needs to specify the appropriate operation in the URL/path sent to the server. A correct implementation of this task, then, will not have a repeat of the socket opening and parsing code 4 times in each of the 4 arithmetic functions, but instead will call a common function, possibly passing in the HTTP get header it should use to send to the CGI script.

Extra Credit: The client side of this exercise could easily and greatly benefit from a test harness to test the arithmetic functions. For extra credit, include unittest code to test the performance of each of the 4 functions. You should test normal cases, and things like cases where negative numbers or 0 are operated on.