**CPSC 261: Lab 9**

**Lab 9 - Network Programming - Sockets**

* [Goals](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab9.html#goals)
* [Introduction](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab9.html#introduction)
* [Overview](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab9.html#overview)
* [The Details](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab9.html#details)
* [Provided Materials](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab9.html#provided)
* [Handing It In](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab9.html#handing)

**Goals**

The goals of this lab are to:

* Give you experience using sockets to communicate between processes across the Internet.

**Introduction**

This lab will help you explore the server side of network programming using sockets. You will be writing the world's simplest web server.

**Overview**

This lab has just one part:

* You will write a very simple web server, that will respond appropriately to only two valid request types and will respond to every other request with an error response.

**The Details**

* Step 1 - In this step you will write a simple web server in a source file named webserver.c. It should listen at a port that you specify on the command line, and respond to requests from web clients (like the one you wrote in lab 8) for the document /index.html with a response as shown below, the document /calculate.html?expression with a response as shown below, and to every any other document requests with an error message.

In this step you will be using the system calls appropriate for server applications rather then client applications. So after establishing a socket, your server will need to execute the listen() system call to prepare the socket for accepting new connections, and then the accept() system call to accept an incoming connection request. The accept() system call blocks the calling thread until an incoming connection request is received. The accept() system call returns a file descriptor for a new socket, one distinct from the original socket that your server created, leaving the original socket available for accepting additional connections.

Once you have a connection to a client program, you will read the HTTP request from the socket, it should look just like the one that your client prepared in lab 8. However, it may have a number of additional header fields. You should accept any incoming header fields, but you should ignore all of them except to ensure that the request is a GET and to read the name of the document.

If the name of the document is /index.html then you should respond with the appropriate header lines, and a small HTML document that includes a message that this is a project for CPSC 261 at UBC. If the name of the document starts with /calculate.html? then you should read the expression that follows the ?, calculate its value, and return it in a web page. Finally, if the name of the document is neither of the two legal documents described above, or if the request is not a GET, then you should respond with an error message. Your success message should look like:

HTTP/1.0 200 OK

Date: Wed, 25 Mar 2014 22:12:51 GMT

Server: Norm's fake web server

Content-Type: text/html

Followed by a blank line, followed by the HTML document.

Of course, you should replace my name with yours!

Your failure message should look like:

HTTP/1.0 400 Bad Request

Date: Wed, 25 Mar 2014 22:12:38 GMT

Server: Norm's fake web server

Content-Type: text/html

Followed by a blank line and optionally an HTML document that you would have displayed to the user.

If you want some examples of error replies, use your webclient to send bad requests to real web servers.

When the request is /index.html you should reply with a *static* document (one that never changes) of your own choosing.

When the request starts /calculate.html?, then what follows the ? will be a simple arithmetic expression of the form

345/5

or

6\*7

These expressions will have only one operator, involve only positive integer constants, and will contain no spaces. You should perform the requested calculation and return a web page that contains the expression that you were asked to calculate and the answer. Sort of like what you get from Google if you ask it for an expression (try it), but you don't have to provide a Javascript calculator in your response. Something like

7\*6=42

appropriately encoded in html, will be sufficient.

If you want a bit of a challenge, and to get a couple of bonus marks, make your web server capable of evaluating any expression involving the usual 4 arithmetic operators and parentheses, like:

44+5\*(63-39)

When your webserver and webclient are both working, they should be able to talk to each other! Test this out.

**Provided Materials**

You should use subversion to check out the lab9 directory from your repository, in the same way that you did for the previous labs.

Your lab9 directory should contain the following files:

* Makefile

The provided Makefile contains instructions for compiling and linking the programs webserver.c.

You will need to create all of the other files. You should be able to reuse your tcp.h and tcp.c files from lab 8.