CS 118: Computer Network Fundamentals

Project 1: Concurrent Web Server Using BSD Sockets

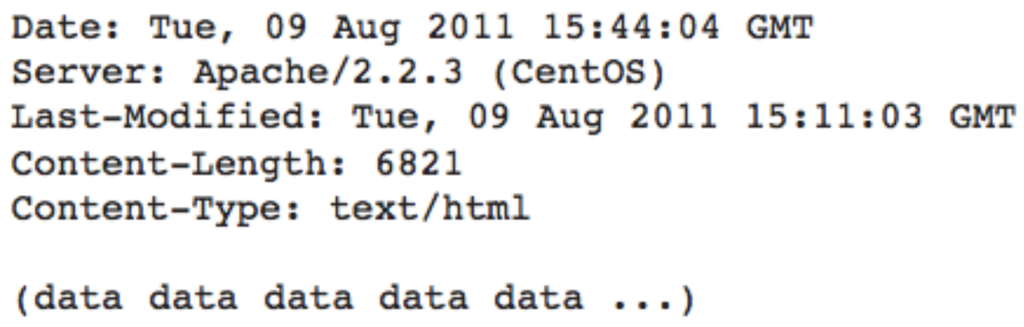
Won Kyu Lee	Justin Young Hoa Lee
UID: 904083134	UID: 204027113
SEAS ID: won	SEAS ID: hoa

High Level Description:

Implementation of Part A of the project involved creating a web server that dumped a response message to the console. To do so, a socket is initialized and the port number is taken in as a parameter. The port number is bound and the server waits for a request. Requests are then dealt with and the HTTP response message and content are written into a buffer that gets returned.

Implementation of Part B of the project involved returning a response directly to the client. The response consists of an HTTP response message and the data from the file being accessed in the server.

To build the HTTP response message, we referenced the following example request message from the textbook on page 106.



```
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data data ...)
```

We wrote functions that would be called to fill in the contents of the HTTP response message. Various functions will be called to populate the data such as the content-length of the file of interest.

To deal with the body of the HTTP request, a series of conditional statements exist to check if the file extension is a .html, .jpg, or .gif file. If there is a match, then the conditional redirects to the corresponding function. Else if the file extension is not valid or the file doesn't exist, a 404 error is displayed.

When there is no 404 error, the size of the file is determined. From then, an arbitrary unit of size is used as a block. Then, the contents of the file are extracted from the beginning until the limit of the file size by writing one block at a time to the socket. The process works similar to paging in virtual memory. These contents are written into a buffer that is pre-posed with the HTTP response message that was constructed earlier.

Problems Faced and Solutions:

Several problems were faced during the design and implementation of the project.

Firstly, writing the functions to populate the data of the HTTP response message was taking a bit longer than expected. For instance, getting the content-length of the file of the interest would require some lower level operations to write the characters into an array to determine the size. Since the project did not have restrictions on using libraries, we ended up using a library *sys/stat.h* which was very easy to implement for returning properties of a file. The *time.h* library was also used on top of the *sys/stat.h* library to make it very straightforward in determining the last-modified property.

In addition, working together as a group added more factors to development. Both of us in the group have worked on projects where the partners send each other project files over e-mail. As this was terribly inefficient and horrible way to write code, we set up a Git repository for this project to synchronize our code and have a revision history. Functions were very modularized so that we could assign development tasks and work on them independently. For instance, populating each of the items in the HTTP response message involved calling a different function that returned the necessary value.

Project Manual:

The project consists of a make file that builds *serverFork.c*. Compilation by running the command *make* should create an object file named *serverFork*. Run the file with a port number as a parameter. For instance, to run the server on port 8080, the command *./serverFork 8080* would be used.

Files that we want to access on the server would be placed in the directory or subdirectory of *serverFork.c*. For instance, if a file is located in *./subdir1/subdir2/test.html* and the server is initialized on port 8080, then *localhost:8080/subir1/subdir2/test.html* would be accessed via a web browser.

As a summary of the workflow to compile and run the project:

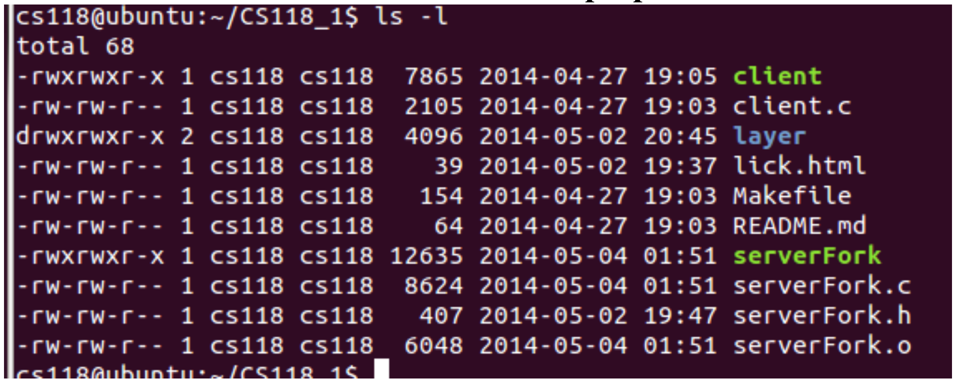
1. *make*
2. *./serverFork* PORT_NUMBER
3. Access *localhost:*PORT_NUMBER/FILE

Sample Output:

A series of screenshots of the terminal will serve as a demonstration of the project. A demonstration of the project involves a file named *lick.html* in the directory where *serverFork.c* is contained. It will be organized as followed:

1. Where the file is located and some of its properties
2. Contents of the file
3. Compilation and running
4. Accessing as client from browser
5. Response message on server

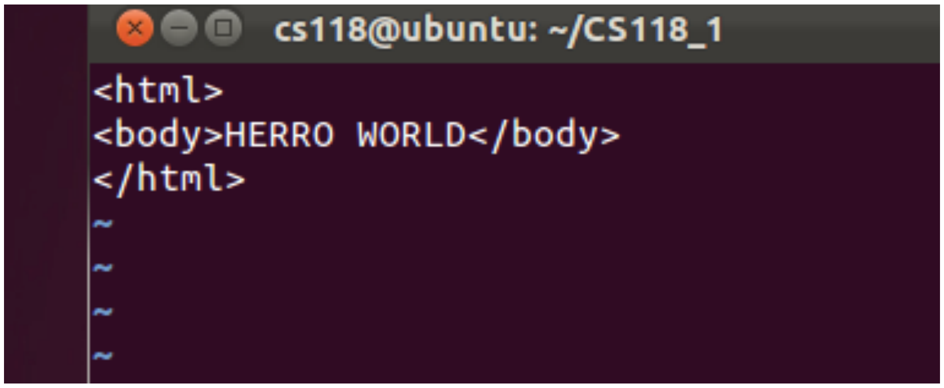
1. Where the file is located and some of its properties



```
cs118@ubuntu:~/CS118_1$ ls -l
total 68
-rwxrwxr-x 1 cs118 cs118  7865 2014-04-27 19:05 client
-rw-rw-r-- 1 cs118 cs118  2105 2014-04-27 19:03 client.c
drwxrwxr-x 2 cs118 cs118  4096 2014-05-02 20:45 layer
-rw-rw-r-- 1 cs118 cs118    39 2014-05-02 19:37 lick.html
-rw-rw-r-- 1 cs118 cs118   154 2014-04-27 19:03 Makefile
-rw-rw-r-- 1 cs118 cs118    64 2014-04-27 19:03 README.md
-rwxrwxr-x 1 cs118 cs118 12635 2014-05-04 01:51 serverFork
-rw-rw-r-- 1 cs118 cs118  8624 2014-05-04 01:51 serverFork.c
-rw-rw-r-- 1 cs118 cs118   407 2014-05-02 19:47 serverFork.h
-rw-rw-r-- 1 cs118 cs118  6048 2014-05-04 01:51 serverFork.o
cs118@ubuntu:~/CS118_1$
```

As shown in the image above, the file *lick.html* is 39 bytes long and the last modified date was on May 2
2014.

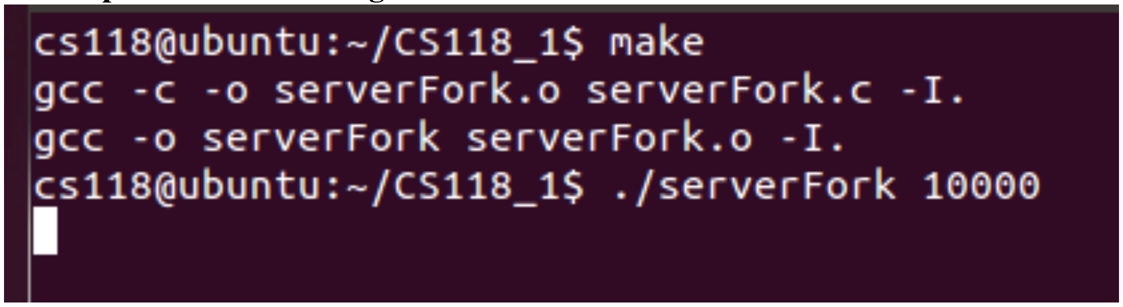
2. Contents of the file



```
cs118@ubuntu: ~/CS118_1
<html>
<body>HERRO WORLD</body>
</html>
~
~
~
~
```

In the image above, *lick.html* contains very basic HTML tags and a body that contains the string "HERRO WORLD".

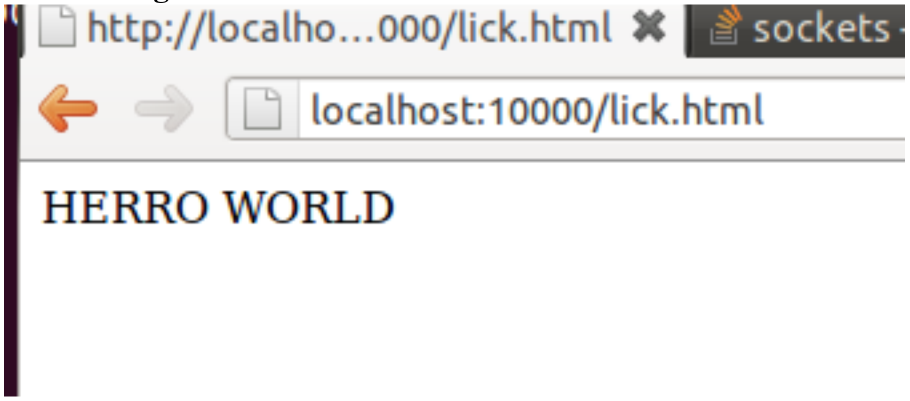
3. Compilation and running



```
cs118@ubuntu:~/CS118_1$ make
gcc -c -o serverFork.o serverFork.c -I.
gcc -o serverFork serverFork.o -I.
cs118@ubuntu:~/CS118_1$ ./serverFork 10000
█
```

serverFork.c is compiled and run per the instructions in the *Project Manual* portion described above. The image above shows the commands and their outputs.

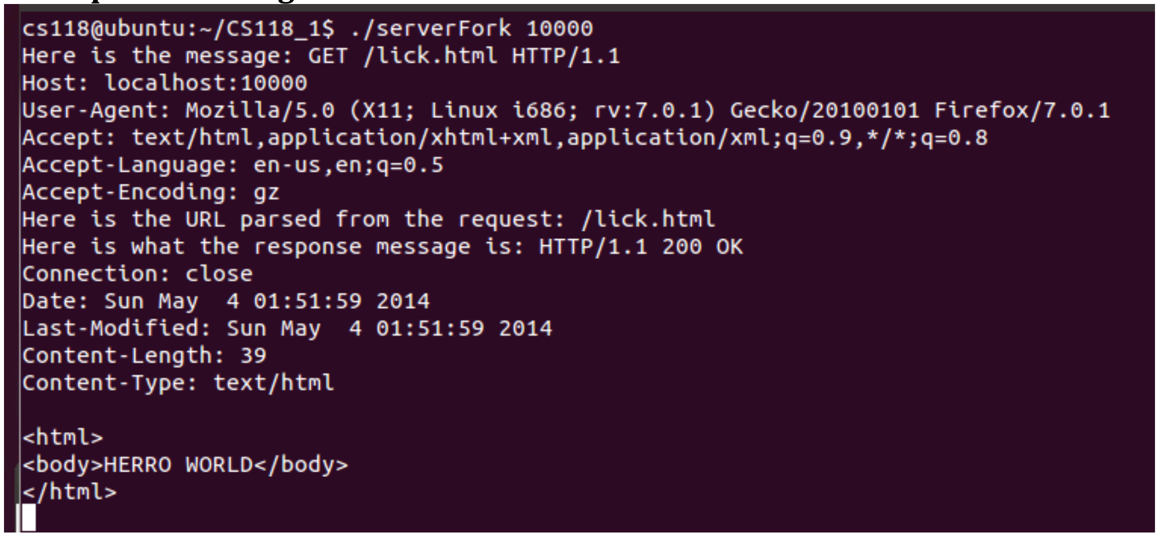
4. Accessing as client from browser



HERRO WORLD

After *serverFork* is run, a browser accesses *lick.html* as described in the *Project Manual* portion described above. The browser prints out the body of *lick.html* as shown in the image above.

5. Response message on server



```
cs118@ubuntu:~/CS118_1$ ./serverFork 10000
Here is the message: GET /lick.html HTTP/1.1
Host: localhost:10000
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gz
Here is the URL parsed from the request: /lick.html
Here is what the response message is: HTTP/1.1 200 OK
Connection: close
Date: Sun May  4 01:51:59 2014
Last-Modified: Sun May  4 01:51:59 2014
Content-Length: 39
Content-Type: text/html

<html>
<body>HERRO WORLD</body>
</html>
```

Meanwhile, an HTTP response message is printed out for accessing *lick.html*. As verified by the image above, the content-length is correctly displayed with a value of 39 and the last modified date is May 4, 2014.