Project 1

Concurrent Web Server using BSD Sockets

CS 118 Computer Network Fundamentals, Fall 2015

Alfred Lucero

ID: 604251044, SEASnet Login: alfred

Khoa Nguyen

ID: 204329993, SEASnet Login: khoanv

Alfred Lucero
Khoa Nguyen
CS118 Project 1 Report

PART A:

For Part A in implementing a simple "Web Server" that dumps request messages to the console, we built off of the example serverFork.c that uses TCP in the internet domain and requires a port number to be passed in as an argument to establish a server that runs forever while forking off a separate process for each connection. We then played around with client.c to cooperate with the server as the client in the internet domain using TCP by passing in a host name and port number. We initially practiced with basic HTTP requests and responses over the Firefox browser already present in the Ubuntu virtual machine. Setting up the VMWare Player and figuring out how to properly use the examples in the terminals in conjunction with the browsers proved to be the only difficulties in this part, and we solved them by understanding client-server interactions and following the usage guidelines in the code.

HIGH-LEVEL DESCRIPTION FOR PART A SERVER DESIGN:

We followed C socket programming principles in first creating a server socket that binds to a port number and listens for client connection requests. Upon accepting a client connection and establishing communication, the server would read in the HTTP request message into a buffer, print out the request into the server console, and write out a response to the client browser. This is straightforward and one would have to do the following in the terminal in the directory of the unzipped files (project1_604251044):

PART A MANUAL:

make // OR gcc –o webserver webserver.c

./webserver 1338 // put in port number of your choice outside of 0-1024 range

Open up Firefox browser and type: localhost:1338 // 127.0.0.1:1338 works as well

EXAMPLE OUTPUT IN THE SERVER CONSOLE:

HTTP Request Message:

GET / HTTP/1.1

Host: localhost:1338

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: gzip, deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Connection: keep-alive.

On a side note, the console may output multiple HTTP request messages for favicon depending on the version of the Firefox browser as it attempts to find a special icon to fit to the left of the URL presented. This is normal and one should not be alarmed by more than one HTTP request message from the local server to the Firefox browser.

PART B:

The objective of Part B is to parse the HTTP request from the client browser, create an HTTP response message of the requested file, and send the response to the client. The difficulties in this part involved formatting the response message to contain the proper fields such as date, content-type, and actual content of the requested HTML file, displaying the contents of JPEG, GIF and HTML files, and handling basic file error statuses or bad requests.

HIGH-LEVEL DESCRIPTION FOR PART B SERVER DESIGN:

First, in order to tackle the formatted response message problem, we organized a struct ResponseMessage that contains the status code, list of header lines, and data content of the varying types of files (.jpg/.jpeg, .gif, .html). We generate a blank one and gradually fill in each of the fields. We parse through the HTTP request message that also outputs to the server console to obtain the file name and type (make use of strtok and strchr functions) and use the stat function to check whether or not the file can be found, returning 404 status errors if necessary. We compose the header lines for content-type, date (make use of strftime function), server, last-modified, content-length, etc. Next, we read in the data of the file into the response message by carefully allocating the proper number of bytes before the fopen/fread functions fill the cstring. Finally, we send the fully formatted HTTP response message fields one by one from the server to the Firefox client browser, and the MIME/html files display properly on webpage.

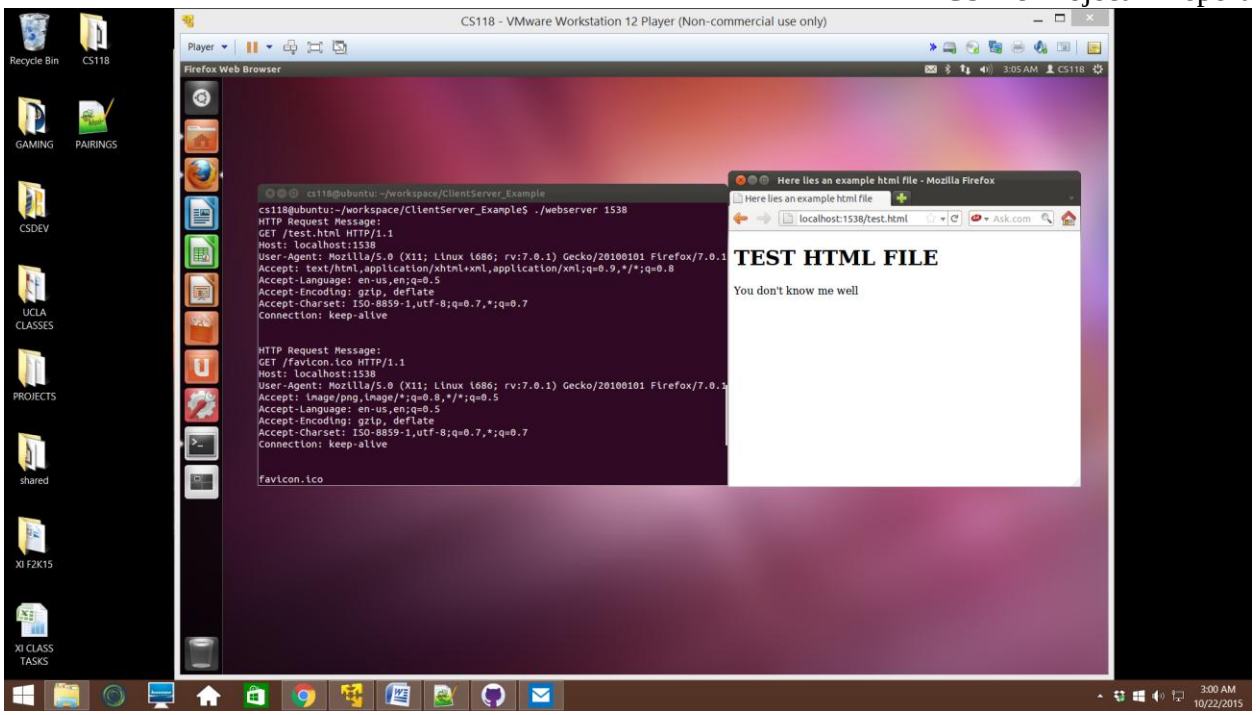
PART B MANUAL WITH EXAMPLE OUTPUTS:

Example test.html, pikachu.jpg, and oreimo.gif files are provided in the directory as well for easier testing. One can see succesful test outputs below for .html/.jpg/.gif files and also error outputs for files that cannot be found or were not specified.

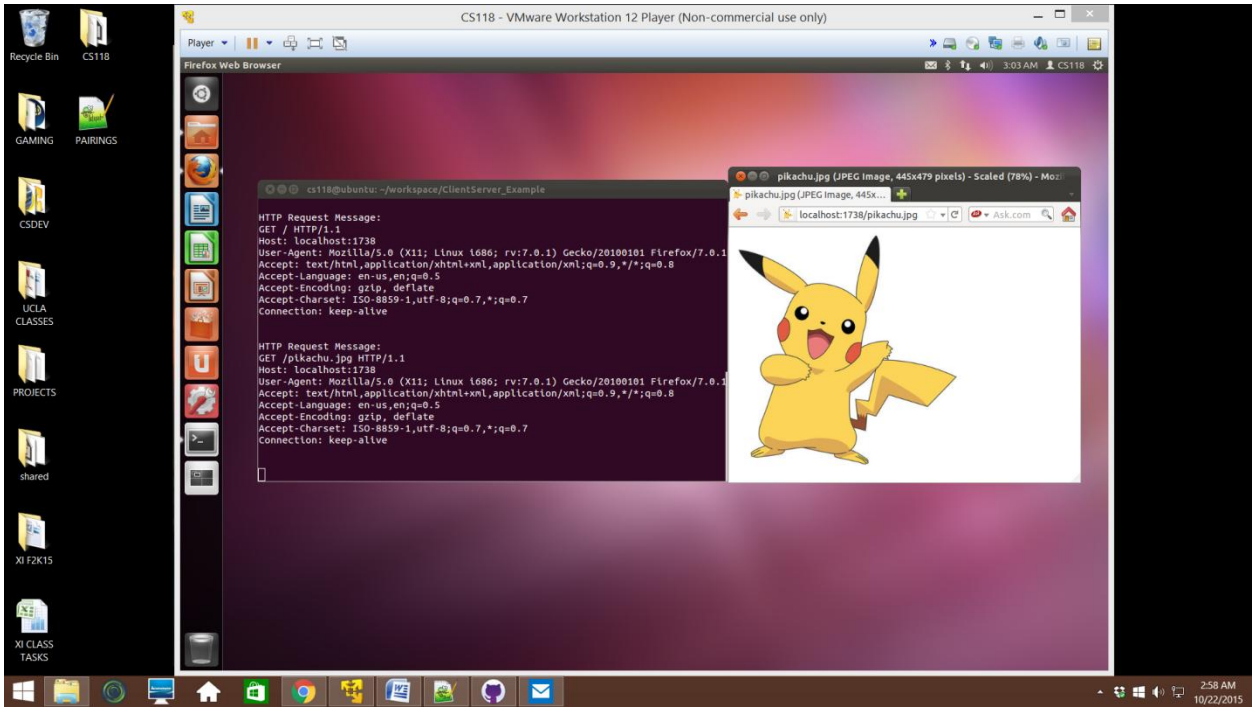
make // in the server file directory project1_604251044 OR gcc –o webserver webserver.c

./webserver port // in server terminal

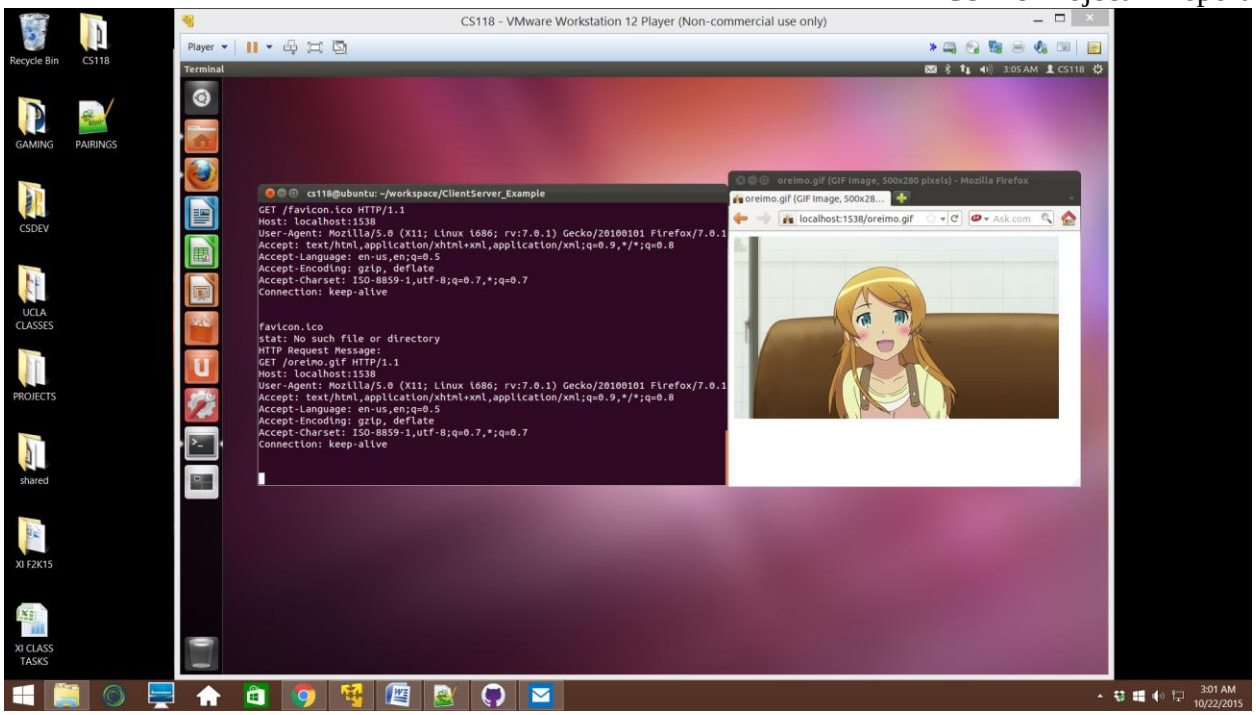
1. localhost:port/test.html // HTML, 200 OK



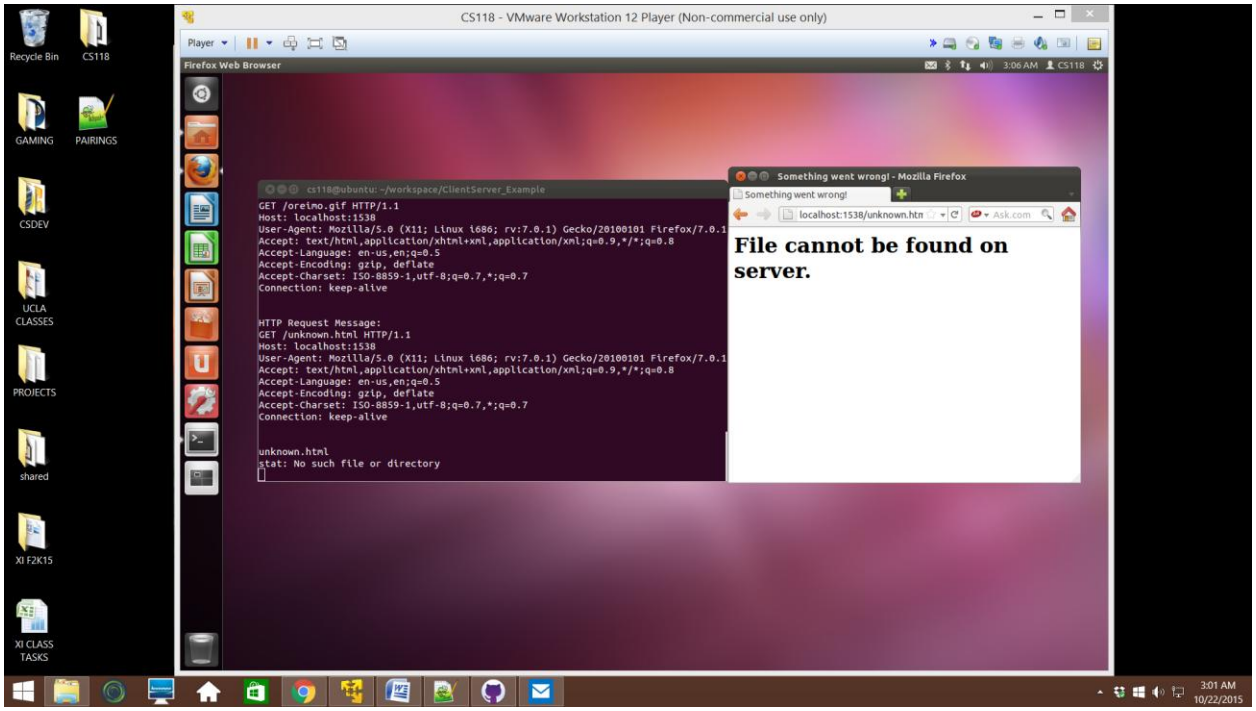
2. localhost:port/pikachu.jpg // JPEG, 200 OK



3. localhost:port/oreimo.gif // GIF, 200 OK



4. localhost:port/unknown.html // FILE NOT FOUND, ERROR 404



5. localhost:port // BAD REQUEST NO FILE SPECIFIED, ERROR 400

Alfred Lucero
Khoa Nguyen
CS118 Project 1 Report