**CS 118: Computer Network Fundamentals**

**Project 1**

**Report**

**Abhijoy Saha (Seas: abhijoy) – 004252758**

**&**

**Kenta Onimura (Seas: kenta) – 804296452**

Design

Our server first creates a server socket and listens to 5 simultaneous connections at most. Then we create a set of sockets (or file descriptors) that will be monitored by the select() function. We first clear the socket list and then add an initial socket representing a server, which we can monitor for new connections.

In a while loop we first block until we receive a request for a new connection on our server socket. When a new connection is requested, we accept it into a new socket and add it to the socket list. We then process the new connection to process the data the client requests.

We proceed to read the client’s HTTP request, which we parse to check for a file name in the request. Then we create an internal stream buffer that performs input operations on the requested file. If we find the requested file, we create the Status Header for the HTTP response by first appending an affirmative (200) message. Then we add the Content-Type field to the HTTP response. We only handle plain, html, gif and jpeg file types. If the file wasn’t found we add to the Status Header an error(404) code and pass a “page not found” error html file to the body of the response. After constructing the HTTP response, we send the response and file data through the new socket to the client. We then remove the new socket from the set of sockets we are monitoring.

Difficulties Faced

Our first difficulty stemmed from attempting to use C to complete our project. Without the string data structure, a lot of our data reading operations required tedious memory allocations and deallocations, initially resulting in segmentation faults that were hard to track down. We decided to solve this problem by backtracking and porting over our server to C++. With the additional libraries, especially the string library, we were able to reduce the number of possible sources of segmentation faults. Our only other difficulty was adding functionality to serve image files. Our fseek approach worked for html and plain text files, but not for image files. We were never able to discern why exactly fseek did not work, but we solved this issue by using file streams to serve the data. By using ifstream, we were able to serve both image and text files with relatively little code.

Running Our Source Code

In the directory where the Makefile and server.cpp files are, run “make clean” to first remove any object and binary files. Then run “make” to create the server binary called “server”. After ensuring the binary is executable with “chmod +x server”, run the server with the command “./server <port>”, where port can be any unused port. To see the HTTP request messages on the terminal, open a browser and search for “localhost:<port>” to see the default message. You can then request a downloaded image/gif/html file by searching “localhost:<port>/<filename>.”

Sample Outputs

From terminal set-up:

$ g++ -o server server.cpp

$ ./server <port-number>

From Client:

Input: http://localhost:2000

Output Client: https://lh6.googleusercontent.com/9zG9ohd1Y-W0MwpD03fpzFkR3-iELF749oNLOWNn4R-2otWa4UXKtp-GiDB5iNfKQRbFdwOAgEcacGsW4YQOInk6vE8iGIBUED9Y3wD4tuKwAkZHFGvIIRXQTkRw05E5-E417ws

Output Terminal:

Here is the message:

GET / HTTP/1.1

Host: localhost:2000

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, deflat

NO FILE ASKED FOR TRANSFER

HTTP Response Message:

HTTP/1.1 200 OK

I got your message

Input: http://localhost:2000/test.txt

Output Client: https://lh5.googleusercontent.com/VtAiW_hKHTU1gI9R_fu6I4HmEZ05b_2Ry6GarkwQVRLAkGcM4D9DIEXpyt8I7qd2TUZdUMcDiwWgkXPDMGZuoRAV8gdVXKGjOkE8-1DZm7OdvhHS8Udaz2eiB_gXkO4vgswmLk4

Output Terminal:

Here is the message:

GET /test.txt HTTP/1.1

Host: localhost:2000

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

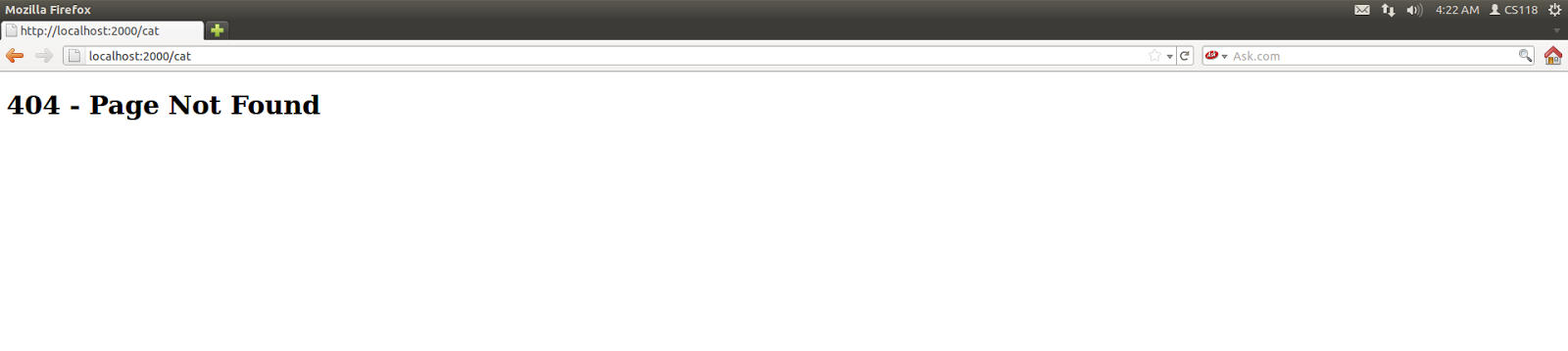
HTTP Response Message:

HTTP/1.1 200 OK

Content-Type: text/plain

I am test.txt

Input: http://localhost:2000/cat

Output Client: 

Output Terminal:

Here is the message:

GET /cat HTTP/1.1

Host: localhost:2000

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, def

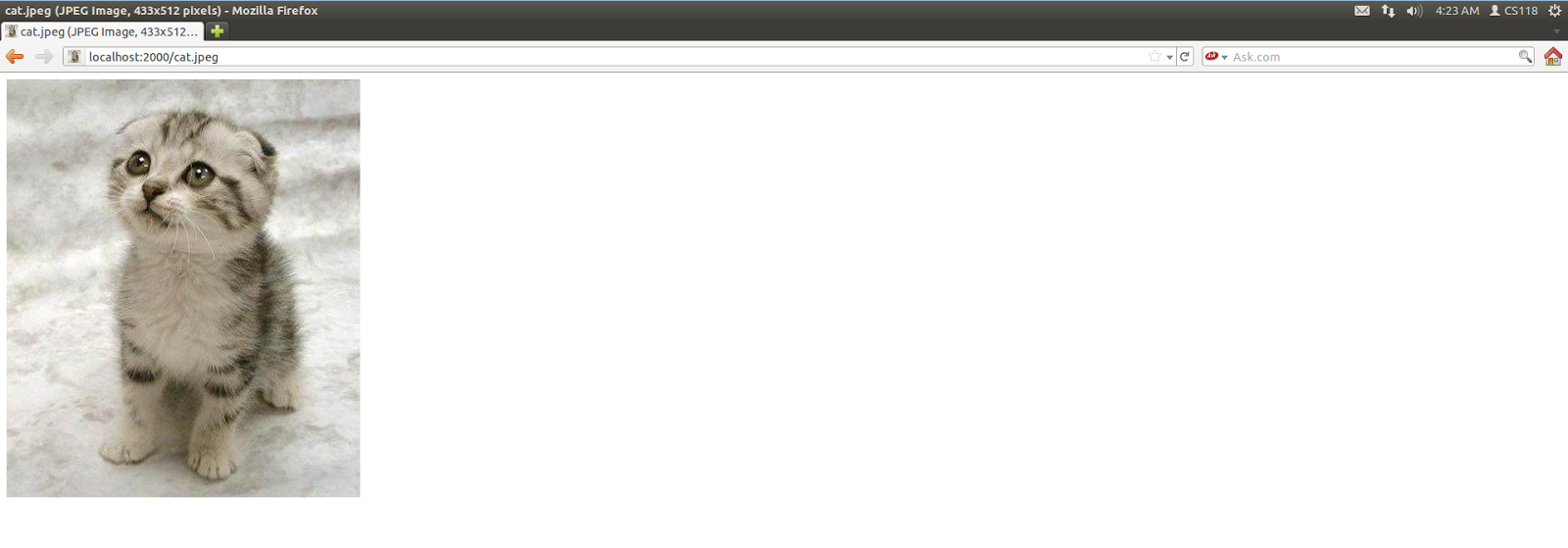
ENTERED FILE NOT FOUND

HTTP Response Message:

HTTP/1.1 404 Not Found

<!DOCTYPE html><html><body><h1>404 - Page Not Found</h1></body></html>

Input: http://localhost:2000/cat.jpeg

Output Client: 

Output Terminal:

GET /cat.jpeg HTTP/1.1

Host: localhost:2000

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

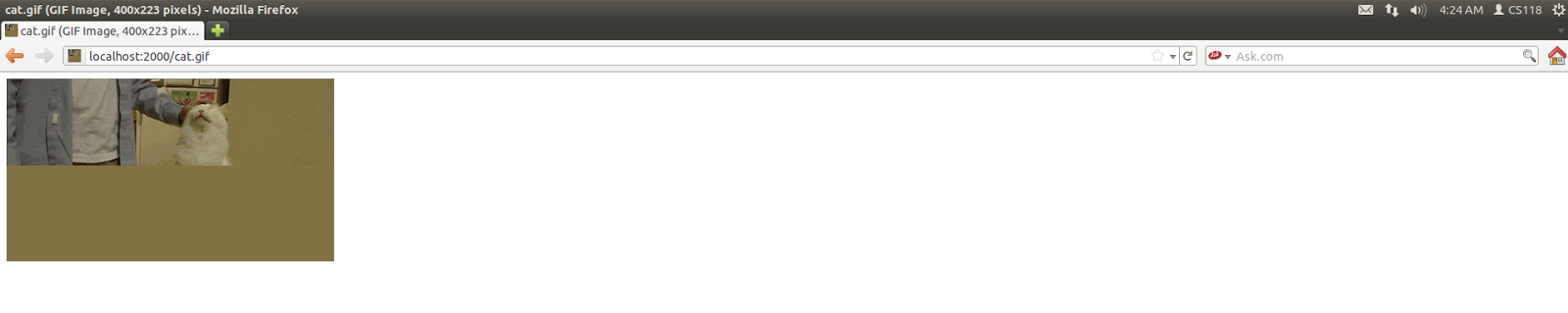
HTTP Response Message:

HTTP/1.1 200 OK

Content-Type: image/jpeg

<cat.jpeg bits start forever>

Input: http://localhost:2000/cat.gif

Output Client: 

Output Terminal:

GET /cat.gif HTTP/1.1

Host: localhost:2000

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

HTTP Response Message:

HTTP/1.1 200 OK

Content-Type: image/gif

<cat.gif bits start forever>