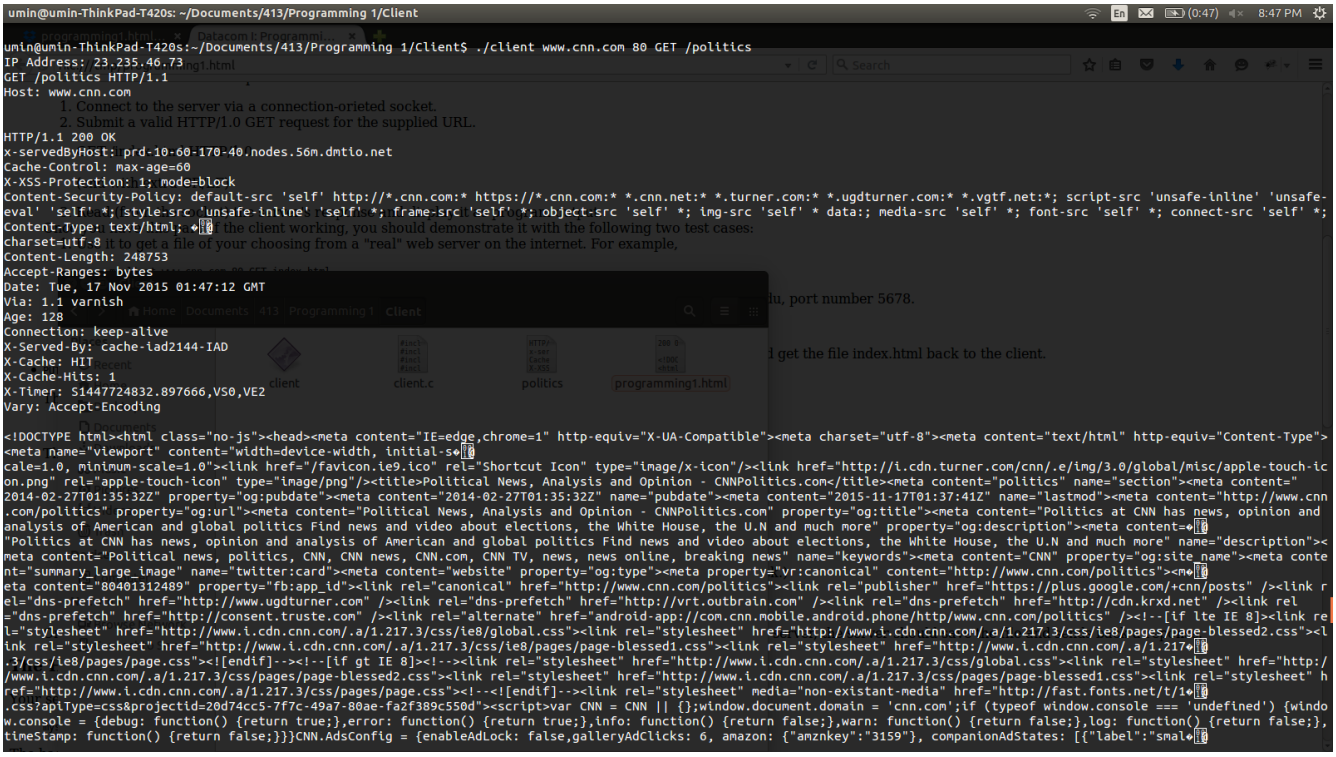
Yoo Min Cha
91.413
Professor Chen
Programming 1

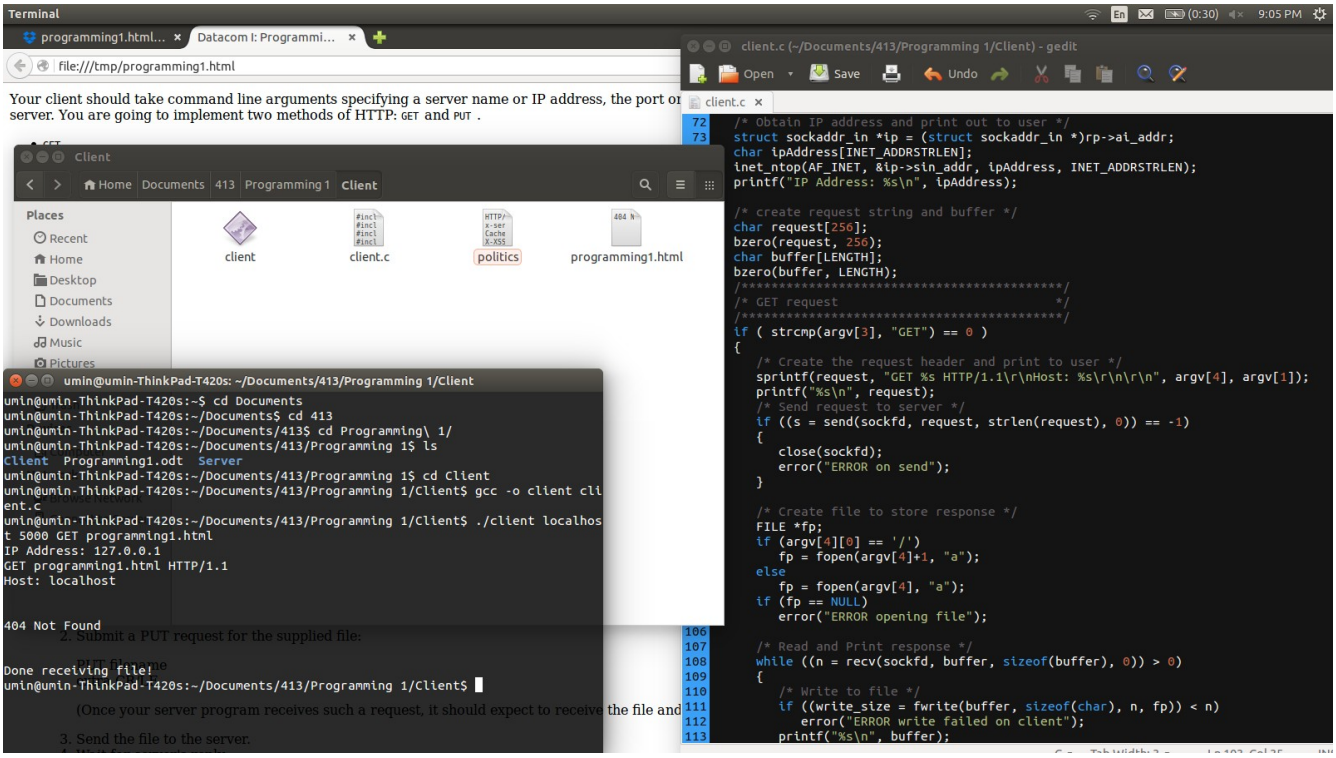
For the HTTP server and client programming assignment, I started building my server and client by creating basic TCP protocol end to end sockets, one being the server and the other being the client. I used the Unix API and utilized the addrinfo structure for my client and used sockaddr_in for my server. I started off by basically just having the server listen for incoming calls on the socket assigned with an ip address and a port number. The client then would try to connect to this server on the localhost and port number. After the server has accepted the connection to the client, the server would create a new file descriptor for the client. After this new file descriptor has been created, I have a two way sequenced reliable connection-based stream between the two sockets. I was able to write from the client and have the server display anything was read in from its socket.

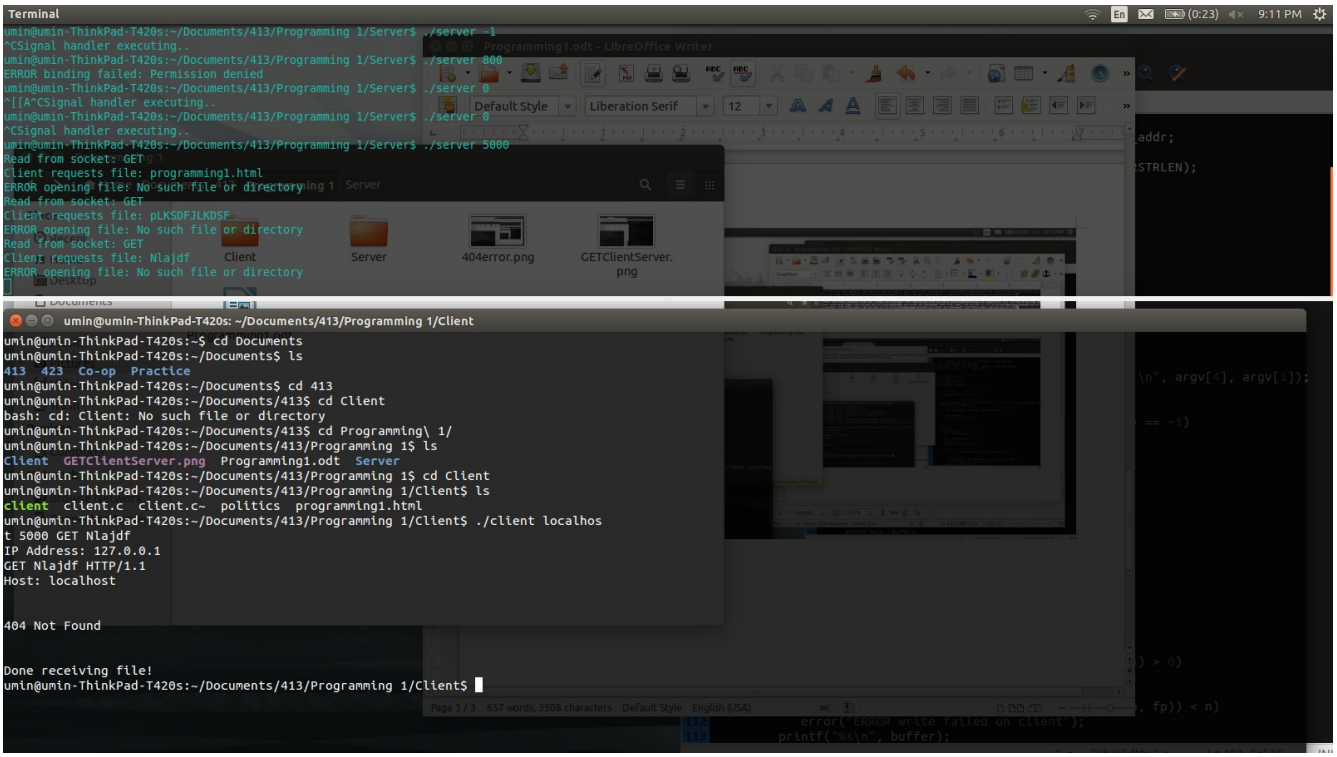
I then added more functionality to the client and server. I started off by working on the PUT requests for both the client and the server. Knowing the name of the file on the client side, I opened a file descriptor to the file and read in from the file while simultaneously writing to the socket which would send the data to the server and then the server would read in from its socket continuously. I decided it would be a good idea for the server to create its own writeable file into which it would store the data coming from the client. After the server had received the entire file and saved it, it would send back a response to the client informing it that the file was successfully transferred. After I got this working I decided to move on to the GET requests for the client and the approach was slightly different. I had to create a new request string using sprintf and then sending the request to the server. I would then read in from its socket after sending the GET request because I knew data would be sent from the server after the request has been sent. I also decided it would be a good idea to store the data coming in to a file so that I could open the file and see if the requested file was the one I was looking for. I tested this on the cnn.com server and requested the index. I got a 404 error stating that the file was not found but it also sent me an html file that worked and loaded up in the browser and proved that the GET file was working. I then had to get the GET request to work for my own server. I did this by opening the requested file it was found and then sending it back to the client. If the file was not found, I sent back a 404 file not found message and if the file was found I sent back a 200 OK message and the file after the message. I then proceeded to making the server to allow multiple connections by forking off each accepted client. My server only allowed a max of 5 clients queued up at the same time. I wasn't able to test the functionality of this because the file transfers were too fast and I could not run more than 5 clients fast enough to test if the server connection queue would fill up.

Test Cases Below

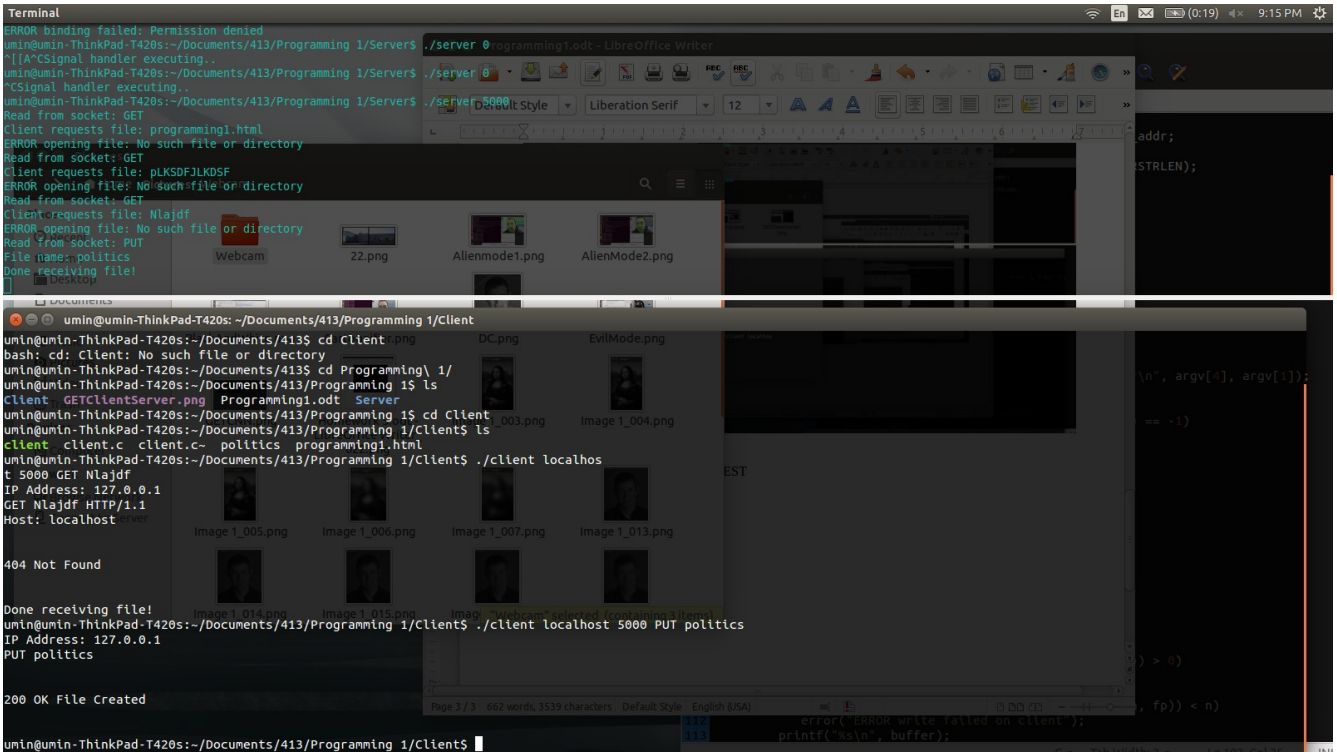
Here is the output from my GET command from client. The file from the output is saved in the Client folder. I was also able to GET the file from the server. I used the programming1.html and stored on the server side. Running ./client localhost PORTNUMBER GET programmin1.html will retrieve the file from the server. Inputting the wrong file name to server will result in a 404 ERROR.


umin@umin-ThinkPad-T420s: ~/Documents/413/Programming 1/Client
umin@umin-ThinkPad-T420s:~/Documents/413/Programming 1/Client$ ./client www.cnn.com 80 GET /politics
IP Address: 23.235.46.73
GET /politics HTTP/1.1
Host: www.cnn.com
HTTP/1.1 200 OK
x-servedByHost: prd-10-60-170-40.nodes.56m.dmtio.net
Cache-Control: max-age=60
X-XSS-Protection: 1; mode=block
Content-Security-Policy: default-src 'self' http://*.cnn.com:* https://*.cnn.com:* *.cnn.net:* *.turner.com:* *.ugdturner.com:* *.vgtf.net:*; script-src 'unsafe-inline' 'unsafe-eval' 'self' *; style-src 'unsafe-inline' 'self' *; frame-src 'self' *; object-src 'self' *; img-src 'self' * data:; media-src 'self' *; font-src 'self' *; connect-src 'self' *;
Content-Type: text/html; ◆
charset=utf-8
Content-Length: 248753
Accept-Ranges: bytes
Date: Tue, 17 Nov 2015 01:47:12 GMT
Via: 1.1 varnish
Age: 128
Connection: keep-alive
X-Served-By: cache-iad2144-IAD
X-Cache: HIT
X-Cache-Hits: 1
X-Timer: S1447724832.897666,VS0,VE2
Vary: Accept-Encoding
<!DOCTYPE html><html class="no-js"><head><meta content="IE=edge,chrome=1" http-equiv="X-UA-Compatible"><meta charset="utf-8"><meta content="text/html" http-equiv="Content-Type"><meta name="viewport" content="width=device-width, initial-s◆
cale=1.0, minimum-scale=1.0"><link href="/favicon.ie9.ico" rel="Shortcut Icon" type="image/x-icon"/><link href="http://i.cdn.turner.com/cnn/.e/img/3.0/global/misc/apple-touch-icon.png" rel="apple-touch-icon" type="image/png"/><title>Political News, Analysis and Opinion - CNNPolitics.com</title><meta content="politics" name="section"><meta content="2014-02-27T01:35:32Z" property="og:pubdate"><meta content="2014-02-27T01:35:32Z" name="pubdate"><meta content="2015-11-17T01:37:41Z" name="lastmod"><meta content="http://www.cnn.com/politics" property="og:url"><meta content="Political News, Analysis and Opinion - CNNPolitics.com" property="og:title"><meta content="Politics at CNN has news, opinion and analysis of American and global politics Find news and video about elections, the White House, the U.N and much more" property="og:description"><meta content=◆
"Politics at CNN has news, opinion and analysis of American and global politics Find news and video about elections, the White House, the U.N and much more" name="description"><meta content="Political news, politics, CNN, CNN news, CNN.com, CNN TV, news, news online, breaking news" name="keywords"><meta content="CNN" property="og:site_name"><meta content="summary_large_image" name="twitter:card"><meta content="website" property="og:type"><meta property="vr:canonical" content="http://www.cnn.com/politics"><m◆
eta content="80401312489" property="fb:app_id"><link rel="canonical" href="http://www.cnn.com/politics"><link rel="publisher" href="https://plus.google.com/+cnn/posts" /><link rel="dns-prefetch" href="http://www.ugdturner.com" /><link rel="dns-prefetch" href="http://vrt.outbrain.com" /><link rel="dns-prefetch" href="http://cdn.krxd.net" /><link rel="dns-prefetch" href="http://consent.truste.com" /><link rel="alternate" href="android-app://com.cnn.mobile.android.phone/http/www.cnn.com/politics" /><!--[if lte IE 8]><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/ie8/global.css"><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/ie8/pages/page-blessed2.css"><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/ie8/pages/page-blessed1.css"><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217◆
.3/css/ie8/pages/page.css"><![endif]--><!--[if gt IE 8]><!--><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/global.css"><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/pages/page-blessed2.css"><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/pages/page-blessed1.css"><link rel="stylesheet" href="http://www.i.cdn.cnn.com/.a/1.217.3/css/pages/page.css"><!--<![endif]--><link rel="stylesheet" media="non-existant-media" href="http://fast.fonts.net/t/1◆
.css?apiType=css&projectid=20d74cc5-7f7c-49a7-80ae-fa2f389c550d"><script>var CNN = CNN || {};window.document.domain = 'cnn.com';if (typeof window.console === 'undefined') {window.console = {debug: function() {return true;},error: function() {return true;},info: function() {return false;},log: function() {return false;},timeStamp: function() {return false;}}}CNN.AdsConfig = {enableAdLock: false,galleryAdClicks: 6, amazon: {"amznkey":"3159"}, companionAdStates: [{"label":"smal◆


Terminal
programming1.html...
Datacom I: Programmi...
file:///tmp/programming1.html
Your client should take command line arguments specifying a server name or IP address, the port of
server. You are going to implement two methods of HTTP: GET and PUT .
Client
Home Documents 413 Programming 1 Client
Places
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
client
client.c
politics
programming1.html
umin@umin-ThinkPad-T420s: ~/Documents/413/Programming 1/Client
umin@umin-ThinkPad-T420s:~$ cd Documents
umin@umin-ThinkPad-T420s:~/Documents$ cd 413
umin@umin-ThinkPad-T420s:~/Documents/413$ cd Programming\ 1/
umin@umin-ThinkPad-T420s:~/Documents/413/Programming 1$ ls
Client Programming1.odt Server
umin@umin-ThinkPad-T420s:~/Documents/413/Programming 1$ cd Client
umin@umin-ThinkPad-T420s:~/Documents/413/Programming 1/Client$ gcc -o client client.c
umin@umin-ThinkPad-T420s:~/Documents/413/Programming 1/Client$ ./client localhost 5000 GET programming1.html
IP Address: 127.0.0.1
GET programming1.html HTTP/1.1
Host: localhost
404 Not Found
Done receiving file!
umin@umin-ThinkPad-T420s:~/Documents/413/Programming 1/Client$
client.c (~/Documents/413/Programming 1/Client) - gedit
Open
Save
Undo
client.c
/* Obtain IP address and print out to user */
struct sockaddr_in *ip = (struct sockaddr_in *)rp->ai_addr;
char ipAddress[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &ip->sin_addr, ipAddress, INET_ADDRSTRLEN);
printf("IP Address: %s\n", ipAddress);
/* create request string and buffer */
char request[256];
bzero(request, 256);
char buffer[LENGTH];
bzero(buffer, LENGTH);
/***/
/* GET request */
/***/
if (strcmp(argv[3], "GET") == 0)
{
/* Create the request header and print to user */
sprintf(request, "GET %s HTTP/1.1\r\nHost: %s\r\n\r\n", argv[4], argv[1]);
printf("%s\n", request);
/* Send request to server */
if ((s = send(sockfd, request, strlen(request), 0)) == -1)
{
close(sockfd);
error("ERROR on send");
}
/* Create file to store response */
FILE *fp;
if (argv[4][0] == '/')
fp = fopen(argv[4]+1, "a");
else
fp = fopen(argv[4], "a");
if (fp == NULL)
error("ERROR opening file");
/* Read and Print response */
while ((n = recv(sockfd, buffer, sizeof(buffer), 0)) > 0)
{
/* Write to file */
if ((write_size = fwrite(buffer, sizeof(char), n, fp)) < n)
error("ERROR write failed on client");
printf("%s\n", buffer);



SERVER AND CLIENT PUT REQUEST



Programming Assignment 1: Programming with Sockets

In this assignment you will be asked to implement an HTTP client and server running a pared down version of HTTP/1.0. You will have to extend the client and server to make use of some of the application level headers we have studied in class. For those of you who already have socket programming experience, the basic part of this assignment will be fairly easy. I am therefore adding an extra credit component for those students who want a more challenging assignment. This project can be completed in either C/C++ or Java.

What to Turn In

When you hand in your programming assignment, you should include:

- A program listing containing in-line documentation. Uncommented code will be heavily penalized.
- A separate (typed) document of a page or so describing the overall program design, a verbal description of ``how it works'', and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
- A separate description of the test cases you ran on your program to convince yourself (and me) that it is indeed correct, and execution traces showing these test being run. Also describe any cases for which your program is known not to work correctly. The test cases you should hand are described below.

The HTTP Client

Your client should take command line arguments specifying a server name or IP address, the port on which to contact the server, the method you use, and the path of the requested object on the server. You are going to implement two methods of HTTP: `GET` and `PUT` .

- `GET`
  The format of the command line is

  ```
  myclient host port_number GET filename
  ```

  The basic client action should proceed as follows:

  1. Connect to the server via a connection-orieted socket.
  2. Submit a valid HTTP/1.0 GET request for the supplied URL.

     GET /index.html HTTP/1.0

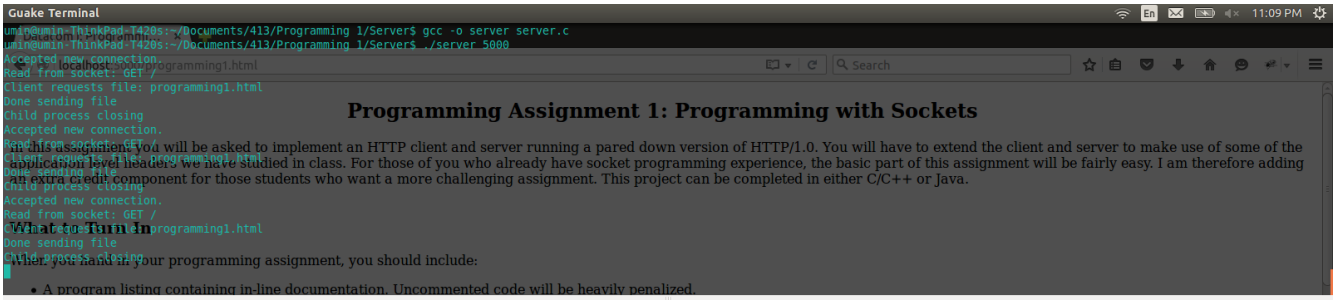
     (end with extra CR/LF)

  3. Read (from the socket) the server's response and display it as program output.

  Once you have this part of the client working, you should demonstrate it with the following two test cases:

  1. Use it to get a file of your choosing from a "real" web server on the internet. For example,

     ```
     myclient www.cnn.com 80 GET index.html
     ```



Programming Assignment 1: Programming with Sockets

In this assignment you will be asked to implement an HTTP client and server running a pared down version of HTTP/1.0. You will have to extend the client and server to make use of some of the application level headers we have studied in class. For those of you who already have socket programming experience, the basic part of this assignment will be fairly easy. I am therefore adding an extra credit component for those students who want a more challenging assignment. This project can be completed in either C/C++ or Java.

What to Turn In

When you hand in your programming assignment, you should include:

- A program listing containing in-line documentation. Uncommented code will be heavily penalized.
- A separate (typed) document of a page or so describing the overall program design, a verbal description of ``how it works'', and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
- A separate description of the test cases you ran on your program to convince yourself (and me) that it is indeed correct, and execution traces showing these test being run. Also describe any cases for which your program is known not to work correctly. The test cases you should hand are described below.

The HTTP Client

Your client should take command line arguments specifying a server name or IP address, the port on which to contact the server, the method you use, and the path of the requested object on the server. You are going to implement two methods of HTTP: `GET` and `PUT` .

- `GET`
  The format of the command line is

  ```
  myclient host port_number GET filename
  ```

  The basic client action should proceed as follows:

  1. Connect to the server via a connection-orieted socket.
  2. Submit a valid HTTP/1.0 GET request for the supplied URL.

     GET /index.html HTTP/1.0

     (end with extra CR/LF)

  3. Read (from the socket) the server's response and display it as program output.

  Once you have this part of the client working, you should demonstrate it with the following two test cases:

  1. Use it to get a file of your choosing from a "real" web server on the internet. For example,

     ```
     myclient www.cnn.com 80 GET index.html
     ```