

NETWORK LAB

Mini Project for Week # 9 & Mid-sem Break

Socket Programming - HTTP Client/Server Application using Concurrent Server

In this assignment you will be asked to implement an HTTP client and server running a pared down version of HTTP/1.1. You will have to extend the client and server to make use of some of the application level headers we have studied in class.

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-oriented 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:

4. Use it to get a file of your choice from a "real" web server on the Internet. For example,

myclient www.ndtv.com 80 GET index.html

5. Use it to get a file from your own server program. For example, your server is running on 172.31.132.x, port number 5678.

myclient 172.31.132.x 5678 GET index.html

This command would result in an HTTP GET request to 172.31.132.x for index.html on port 5678, and get the file index.html back to the client.

- **PUT**

The format of the command line is -

myclient host port_number PUT filename

The basic client action should proceed as follows:

1. Connect to the server via a connection-oriented socket.
2. Submit a PUT request for the supplied file:
PUT filename
extra CR/LF
(Once your server program receives such a request, it should expect to receive the file and save it to disk.)
3. Send the file to the server.
4. Wait for server's reply.

Once you have this part of the client working, you should test it with your own server: send out a file to your server, the server should save the file and send back a response.

The HTTP Server

Your server should take command line arguments specifying a port number. For example, `myserver 5678`

The basic server action should proceed as follows:

1. Initialize the server.
2. Wait for a client connection on the port number specified by command line argument.
3. When a client connection is accepted, read the HTTP request.
4. Construct a valid HTTP response including status line, any headers you feel are appropriate, and, of course, the requested file in the response body.

For GET , if the server receives the "GET index.html HTTP/1.0" request, it sends out "200 OK" to the client, followed by the file index.html. If the requested file doesn't exist, the server sends out "404 Not Found" response to the client.

For PUT , if the server receives the "PUT test.txt" request, it will save the file as test.txt. If the received file from client is successfully created, the server sends back a "200 OK File Created" response to the client.

5. Close the client connection and loop back to wait for the next client connection to arrive.

Notice that your server will be structured around an infinite loop. That means that you must interrupt your server with a termination signal to stop it. Make sure your server code shuts down gracefully when terminated. That means closing any open sockets, freeing allocated memory, etc.

Once you get your server working, demonstrate it with the following two test cases:

First, use an ordinary browser such as Firefox or Chrome to get a *html* file from your server. For example, your server is running at host `netlab10.csed.mnnit.ac.in` on port number 5678, and there is a file `index.html` in the current directory. In the URL box of the web browser, type in `172.31.132.x:5678/index.html`, the browser should fetch the file and display it.

Second, use your own client to get a file.

Third, use your own client to put a file.

Make your server to be a concurrent server - that is a server that waits on the welcoming socket and then creates a new thread or process to handle the incoming request.

(Initially your server code may be written as an iterative server program, and once you have done with the testing using the client, the code can be rewritten as a concurrent server program).

Programming Notes:

Here are a few tips/thoughts to help you with the assignment:

- You must chose a server port number greater than 1023 (to be safe, choose a server port number larger than 5000).
- It is strongly suggested that everyone begin by writing a client and getting its test cases to work. Then write a sequential server and test it with your client. Finally, write a concurrent server, if you want to do the extra credit portion of the assignment.
- In writing your code, make sure to check for an error return from your system calls or method invocations, and display an appropriate message.
- Most of you will be running both the client and server on the same machine (e.g., by starting up the server and running it in the background, and then starting the client) under Linux. Recall the use of the *ampersand* to start a process in the background. If you need to kill your server after you have started it, you can use the Linux *kill* command. Use the Linux *ps* command to find the process id of your server.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port IDs, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use. On Linux systems, you can run the command "*netstat*" to see which port numbers are currently assigned.

Note:

This assignment has to be completed during lab sessions of Week 9 and the mid-sem break.