Programming Assignment 2: Webserver

P538 Computer Networks

Assigned: February 6, 2018

Due: February 21, 2018 11:59 pm

Instructions:

- You may discuss with your classmate, but please submit your own individual assignment.
- Please submit your code via Canavs
- If you have debug statements, please remove them or comment them out before submitting.
- In addition to C source code files, supply a makefile and address all compilation errors.
- Review Section 2.7: Socket Programming: Creating Network Applications

The purpose of this assignment is for you to learn the basics of socket programming for TCP connections: how to create a socket, bind it to a specific address and port, as well as send and receive a HTTP packet. You will also learn some basics of HTTP header format.

Part I: Webserver (20 pts)

Develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client. Your server should support non-persistent and persistent connections.

Running your Webserver

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80.

Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

Part II: Multi-threaded Web Server (10 points)

Currently, the web server handles only one HTTP request at a time. Using Pthreads, implement a multithreaded server, that is capable of serving multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair.

Meaning many threads with different port number.

http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

Part II. Web Client (20 pts)

Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method. Your client should request a basic text file, .txt. Once that file has been received, the client should print the file to stdout.

The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, whether the connection is non-persistent or persistent, and the path at which the requested object is stored at the server. The following is an input command format to run the client.

```
client server_host server_port connection_type filename.txt
```

Part III. Connection-less, Unreliable client and server (20 pts)

Write a new version of your client and server applications using UDP sockets. Your client should implement a HTTP Get request and record the number of bytes that it receives. The server should process the Get and indicate when the last byte of the file has been transmitted.

Part IV: Writeup (30 pts)

- Test your client and server applications on the CS machines. Compare and contrast the time that it takes to request and receive one to ten 1 MB text files using non-persistent and persistent connections. Explain the trends that you observe. What is the expected RTT in this environment? Does the time taken to service multiple requests grow linearly? Why or Why not?
- Repeat the tests above with the Multi-threaded server. Compare and contrast these results with the single process server.
- Using your connection-less client and server, how long does it take to request and receive a 1 MB file? How does this time compare to the times for persistent and non-persistent connections?Do you experience packet loss when using your UDP client and server applications? If so, when does loss occur?