Programming Assignment: Webserver

P538 Computer Networks

Assigned: 9/6/2015

Due: 9/20/2015 at 11:59 pm

Instructions:

- You may discuss with your classmate, but please submit your own individual assignment.
- Please submit your code via Canvas
- If you have debug statements, please remove them or comment them out before submitting.
- Readability is an important part of your programming assignment. To aide readability of your code, you should modularize your code by decomposing the programming tasks into functions. Do not implement all code in main (). You should also comment all code. Each function should contain comments that describe its function. There should be comments within the function to describe the functionality that is being implemented.
- 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.

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

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 servers should also implement the response "400: Bad Request", which indicates that the request message was not understood by the server. 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. Web Client

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. Your client must implement the Get and use the "Close" option to indicate that the connection is non-persistent. 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

The connection type may be specified as "np" for non-persistent and "p" for persistent. For non-persistent connections, filename.txt indicates the name of the file that will be retrieved from the server. When connection type is specified to be persistent, filename.txt is assumed to be a file that contains a list of filenames that will be retrieved from the server. The format of filename.txt for the latter case is one filename per line.

Part III. Connection-less, Unreliable client and server

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: Multi-threaded Server

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.

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

Part V: Write-up

Your write-up should include an analysis of the time that it takes for your client to receive requested files in various scenarios. You should use the "gettimeofday" function that is included in the standard c library <sys/time.h> to take time measurements. Gettimeofday provides timing data at the granularity of microseconds, but note that you may need to check the resolution of the clock at the operating system to ensure that microseconds is achievable. In addition to timing getting timing data within the code, you should use Wireshark to provide evidence of packets being transmitted between your client and server. Your write-up should include Wireshark screenshots to support your analysis.

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?

Using your connection 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?

Complete the same analysis for the multi-threaded server. In addition, your analysis should discuss the performance received using the UDP client and server applications. Do you experience packet loss when using your UDP client and server applications? If so, when does loss occur?