Simple HTTP Server Lab

CS536

Due: September 23, 2018, 11:59pm

1 Objective

In this lab you will learn the basics of socket programming for TCP connections in Python. Key takeaways:

1. how to create a socket
2. how to bind it to a specific address and port
3. how to send and receive a HTTP packet

2 Getting Started

1. Socket Programming
   - `https://www.geeksforgeeks.org/socket-programming-python/`
2. HTTP Protocols
   - `https://tools.ietf.org/html/rfc1945#section-8`
   - `https://tools.ietf.org/html/rfc1945#section-6`
   - `https://www.tutorialspoint.com/http/http_responses.htm`
   - `https://www.tutorialspoint.com/http/http_requests.htm`

3 Implementation

- Programming Environment: Your code should be able to compile and run correctly on the XINU machines (xinu1.cs.purdue.edu, xinu2.cs.purdue.edu,...) To SSH into those machine, in terminal type *ssh {your purdue username}@xinu{1-10}.cs.purdue.edu*
- Web Server Requirements: (Your main task is to create a multithreaded web server.)
  - You will 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 as 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. If a file is present but the proper permissions are not set, a permission denied error is returned. A file should only be transferred if public has read access for the file.

- You should also note that web servers typically translate GET / to GET /index.html. That is, index.html is assumed to be the filename if no explicit filename is present. When you type a URL into a web browser, it will retrieve the contents of the file. If the file is of type text/html , it will parse the html for embedded links (such as images) and then make separate connections to the web server to retrieve the embedded files. If a web page contains 4 images, a total of five separate connections will be made to the web server to retrieve the html and the four image files. Note that the previous discussion assumes the HTTP/1.0 protocol which is what you will be supporting in this first assignment
- Your server should also provide support for the HEAD commands. The HEAD method asks for a response identical to that of a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
- To run your server, the command should allow user to select the port to operate on. Ex: *python server.py 4567*
    - At a high level, your web server will be structured somewhat like the following:
        - Forever loop :
        - Listen for connection
        - Accept new connection from incoming client
        - Check if the client has made too many requests.
        - Parse HTTP request
        - Ensure well-formed request (return error otherwise)
        - Check if the IP is registered in the database
        - Determine if target file exists and if permissions are set properly (return error otherwise) Transmit contents of file to connect
        - Close the connection
- For this assignment, you will need to support enough of the HTTP protocol to allow an existing web browser (Firefox, Safari or Chrome) to connect to your web server and retrieve the contents of sample pages from your server. (Of course, this will require that you copy the appropriate files to your servers document directory - please name the server document directory as Upload). We will be mainly checking for txt files and images
- Currently, the web server handles only one HTTP request at a time. 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.
- Now we have a basic functioning http server, we would like to add support for cookies. Create a database (Could be just a python dictionary in this case) to keep track each new browser visited your server. For each browser keep track of number of times of visits, and last visiting time. Remember to use lock, since your server is multithreaded. To demonstrate this, write out the database to a CSV or json file when you close your server. Load the database each time when you start the server.

- Our web server now has the capabilities to generate insights from visitors. Next we want to protect our server from denial of service attack. If a user has sent over 100 requests over the past minute, we will ban this IP address. The memory for banned IPs can be refreshed each time the server restart for demonstration purpose.
- 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 or HEAD method. Your client should also support a DOS mode, that rapidly send requests to the server. The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, the path at which the requested object is stored at the server and whether it is a GET or HEAD command, optional DOS mode, and number of requests to send in DOS mode. The following is an input command format to run the client.

  Ex: *client.py serverHost serverPort filename command(GET or HEAD) -d(optional) 200 (number of requests)*
- The client should save the meta data or the data in the folder named as Download with the same name as specified in the command line argument.

4 Submission and grading

1. You will be submitting your assignment on blackboard. Your submission directory should contain:

   (a) All source files

   (b) A README file that contains instruction to run your code to demonstrate all the features required

2. Note:

   (a) Please document any reasonable assumptions you make or information in this file, e.g., if any parts of your assignment are incomplete.

   (b) Please do not use python built in http services.

   (c) Questions about the assignment should be posted on Piazza.