CSCI 440: Computer Networks
Homework 1
Fall 2017

Summary:
We implement a very simple web server in Python or Java or C/C++.

Objectives:

- Practice basic UNIX-style socket programming
- Learn about the HTTP protocol

Background

- Section 2.2 in Kurose and Ross (Computer Networks) details the basics of the HTTP protocol, with section 2.2.3 highlighting the HTTP message format for requests and responses.
- Section 2.7 in Kurose and Ross (Computer Networks) demonstrates how to use Python for socket programming, with 2.7.2 highlighting the TCP protocol you will use.

Collaboration:
You may complete this assignment in pairs.

Introduction

In this lab, you will implement and test a very simple web server using the hypertext transport protocol (HTTP), a text-based application-layer protocol. The basic outline is as follows. Your server will establish a listening socket and wait for connections in an infinite loop (so that it can serve as many as come in while it is waiting). It will then need to accept a connection, then receive and parse the HTTP request. If the request is valid, the server should read the file from the file system and send the file contents (in small chunks, say 2 kilobytes) to the requestor preceded by the appropriate response header lines. If the request is not valid, a 400 error code should be sent instead. If there is some other error reading the file, a 404 error code should be sent.

Implementation

Create a file called wwwserv.py (or wwwserver.java or wwwserver.c). The port your server listens on should be a clearly identifiable variable; its value should be greater than 5000, where users have rights to establish port listeners. You will want to make it a variable

because if you kill the server, the port will remain "in use" for a short while to ensure that no stray requests for the port filter in.
After creating, binding, and listening on the socket, your server body should act like the following:

```
while True
    Establish connection with a client
    Try
        Read a request
        If the request is invalid
            Send a 400 status code, close the connection, and continue
        Open the requested file (strip the leading '/' for a relative path)
        Send a single response header to the client
        Send the file to the client in small chunks
        Close the file and connection
    Handle IO Error (sending a 404 status code and closing the connection)
```

In lieu of a proper logging system, your server should report simple messages to the console as it takes these steps.

Testing

Place a simple HTML file, such as example.html in the same directory as your wwwserv.*. Requests to your web server for files will be relative to the server's directory. For example, if you used port 8765 and were running your program on your local machine (localhost or 127.0.0.1), you can request the file via a web browser with the url http://localhost:8765/example.html or using **telnet**(1), as demonstrated in Kurose and Ross. You can even make such a telnet-based request to a "real" web server, as in the following transcript, which you can try yourself.

```
CSCI-C1MRW0T6H3QK:Project mountrouidoux$ telnet people.cofc.edu 80
Trying 153.9.205.24...
Connected to people.cofc.edu.
Escape character is '^]'.
GET mountrouidoux/CSIS490/index.html
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.7 (Ubuntu) Server at bexley.cougars.int Port 80</address>
</body></html>
Connection closed by foreign host.
CSCI-C1MRW0T6H3QK:Project mountrouidoux$
```

Questions

1. What happens if you try to make a second connection to your server *while* it is still handling another connection?

   For example, make one telnet connection to your server, but don't issue a request yet. Make another telnet connection. If it succeeds, issue a GET request. Then go back to your original connection and issue a GET request.

   What happened? Why?

2. Note that we use the program's directory to implicitly specify where the web server should look for files. Does this guarantee that a malicious web client cannot access files outside of the web server's directory? If so, explain why. If not, give an example of a request a client could make to obtain an unauthorized file.

What to turn in

- Your Python, Java, or C/C++ file, **with descriptive, professional comments.**
- A single PDF containing (merged)
  - A transcript of telnet sessions
    - requesting a short file from your server,
    - making an invalid request, and
    - requesting a non-existent file.
  - A transcript of your server starting and handling these requests
  - Your answers to the questions

Submissions missing the PDF or files in any other formats will not be graded.

Acknowledgments

This lab was adopted from:
http://www.cs.grinnell.edu/~weinman/courses/CSC364/2014S/labs/http-server.html