CS433/533 Homework Assignment Three (Part 1): HTTP Protocol and Web Server

This assignment (Part 1) gives you a chance to better understand HTTP design, including its protocol and the processing at the server. In Part 2, we will extend Part 1 to give you a chance to become more familiar with high performance server design, covering topics including threads, synchronization, wait/notify (monitor), asynchronous I/O, and benchmarking.

Due:

- Part 1: 11:59 PM, Thursday Oct. 11, 2018
- FAQ
- rubric

Protocol

The server that we will design is a simplified version of HTTP 1.0. The most basic application message, encoded in ASCII, from the client to the server is:

```
GET <URL> HTTP/1.0
Host: <ServerName>
CRLF
```

where CRLF is carriage return and line feed, representing an empty line. This request asks for the file stored at location <DocRootServerName>/<URL>, where DocRootServerName is the root directory for the requested server name.

For example, if <DocRooServerNamet>=/tmp/mydoc, and <URL> is /file1.html, the server will return the file /tmp/mydoc/file1.html, if it exists. If the request does not specify the Host header, the server returns the first server (virtual host) configured; see below.

The basic reply message from the server to the client, encoded in ASCII, is:

```
HTTP/1.0 <StatusCode> <message>
Date: <date>
Server: <your server name>
Content-Type: text/html
Content-Length: <LengthOfFile>
CRLF
<file content>
```

CRLF again represents an empty line. If the file is found and readable, the returned <status code> is 200 and you can give a message such as OK. Otherwise, please give an error code of 400. If you are curious about HTTP error codes, you can see http://www.ietf.org/rfc/rfc1945.txt. You can use Java File class to obtain file size.

Part 1a: Simple Client

Your test client should be multi-threaded. The client can generate test requests to the server with the following command line:

```
%java SHTTPTestClient -server <server> -servname <server name> -port <server port> -parallel <# of threads> -files <file name> -T <time of tes
```

In a typical deployment, we do not need to specify both server and servname. For example, in a typical setting, a Web hosting server with IP1 may host multiple virtual hosts named www.vh1.com, www.vh2.com. All of these DNS names will resolve to the same IP1. But for our testing, since the zoo machine typically has only a single DNS name, we add the servname switch to specify the virtual host.

The <file name> is the name of a file that contains a list of files to be requested. For example, a file may look like the following:

```
file1.html
file2.html
file3.html
file1.html
```

Then each thread of the client will request file1.html, then file2.html, then file3.html, and then file1.html. The thread then repeats the sequence. The client simply discards the received reply. The client stops after <time of test in seconds>. The client should print out the total transaction throughput (# files finished downloading by all threads, averaged over per second), data rate throughput (number bytes received, averaged over per second), and the average of wait time (i.e., time from issuing request to getting first data). Think about how to collect statistics from multiple threads.

Part 1b: Sequential and Per-thread HTTP Servers

In class we will cover multiple approaches to implementing network servers. In Part 1, you will need to implement only the sequential server an the per-thread server.

You can feel free to reuse the example code provided in class.

When your server executes, it must support the following:

- Configuration: Your server must support a configuration file, which we follow the Apache configuration style (http://httpd.apache.org/docs/2.4/vhosts/examples.html. Note that we implement a single server name, not multiple, as the Apache example configuration shows. We start a server by reading a configuration file:

  ```
  %java <servername> -config <config_file_name>
  ```

  The basic configuration parameter is listening port:

  ```
  Listen <port such as 6789>
  ```

  A configuration file should also contain one or more virtual hosts shown below. We use the same format as the Apache, but your server ignores the *:6789 part.

  ```
  <VirtualHost *:6789>
    DocumentRoot <root dir>
    ServerName <server name>
  <VirtualHost>
  ```

We recommend that you consider a hash map in your program to implement configurations.

An example configuration file is httpd.conf.

- HTTP Methods: Your server must support HTTP 1.0 (http://www.w3.org/Protocols/HTTP/1.0/spec.html) GET method.

- Headers: Your server must send the Last-Modified header and understand the If-Modified-Since header from client. This means that you will need to parse date format. For this assignment, we use the rfc1123-date format. Your server also needs to understand the User-Agent header. For other headers, your server can skip.

- URL Mapping: If the URL ends with / without specifying a file name, your server should return index.html if it exists; otherwise it will return Not Found. If the request is for DocumentRoot without specifying a file name and the User-Agent header indicates that the request is from a mobile handset (e.g., it should at least detect iphone by detecting iPhone in the User-Agent string), it should return index_m.html, if it exists; index.html next, and then Not Found. It is important to note that URL mapping is a common component for security attack. For example, one could include .. in the path to try to download files outside the document root. In this assignment, if a URL includes .. as a component, it is an invalid URL.

- Caching: Your server needs to include a basic caching mechanism to speedup handling of requests for static files. The cache is a simple Java Map, with key being the file and content the whole file in an array. Before reading a file from disk, the server checks whether it is already cached. Think: how to handle multiple threads reading and adding to the Map.

  The cache size can be specified in the configuration file:

  ```
  CacheSize <cache size in KBytes>
  ```

  To simplify your server, there is no cache replacement; i.e., when the cache is full, no addition to the cache.

- Dynamic content using CGI: Your server needs to check if a mapped file is executable. If so, it should execute the file and relay the results back to clients. Our assignment only handles the case that the input to the external program is from GET. Please see Java ProcessBuilder on how to start set environment variables and start a dynamic process. The example of the doc can be helpful. You will need to read RFC 3875 to set the right environment variables. You will need to write a dynamic CGI program to test your invocation.

- Heatbeat monitoring: Your server needs to implement a heartbeat monitoring URL service to integrate with a load balancer (e.g., Amazon Load Balancer we covered in class). In particular, a load balancer may query a virtual URL (i.e., no mapped file) named `load` (i.e., with request GET /load HTTP/1.0). If the server is willing to accept new connections, it should return status code 200; otherwise, it returns code 503 to indicate overloading. Your software design should follow a "plugin" design, at run time, of different algorithms to compute overloading conditions. In particular, the monitor class file name can be specified in the configuration file.

  ```
  Minotor <MyCoolMonitorClassName>
  ```

  Please describe a particular design and implement it.

Submission

- **Please submit using canvas. Please include README to tell the TA the directory structure. Please generate a single jar file containing all of your files.**

References

- Book
  - Java Network Programming (4th ed.) by Elliotte Harold is a good reference book on Java network programming. Yale library has electronic version of this book (you need Yale IP address to gain access). Please try to search Orbis and follow the link. The examples codes can be found here.
- General java information:
  - Java book
    - An overall very good book on Java is **Thinking in Java (4th Edition)** by Bruce Eckel. The book web site is here.