18-441/741: Computer Networks

Project 3: HTTP Pseudo Streaming

Deadline: 11:59pm EDT, Apr. 25, 2021

Questions? Ask on Piazza!

1. HTTP Server

An HTTP server, often known as a web server, provides content needed by web browsers. The content could be an HTML file, a plain text file, an image, an audio file, a video, or just a binary file. In a nutshell, our HTTP server will act as a simple file server. It will take a resource identifier (URI) and locate the content for the URI in a directory relative to a content root. Then, it will generate a HTTP response message which contains information about the content (headers), and the content itself (body).

2. HTTP Protocol

The Hypertext Transfer Protocol is used to facilitate transfer of hypermedia documents (text, contents, links). The current protocol is based on RFC 2616 (http://www.w3.org/Protocols/rfc2616/rfc2616.html). Please familiarize yourself with the following sections. In the documentation, the server you are implementing in this project is referred to as the "origin server."

- HTTP Message / Request / Response format (4,5,6)
- Status Code
  - `200 OK` (10.2.1)
  - `206 Partial Content` (10.2.7)
  - `404 Not Found` (10.4.5)
  - `500 Internal Server Error` (10.5.1)
- Method
  - `GET` (9.3)
- Header
  - `Accept-Ranges` (14.5)
  - `Connection` (14.10)
  - `Content-Length` (14.13)
  - `Content-Range` (14.16)
  - `Content-Type` (14.17)
  - `Date` (14.18)
  - `Last-Modified` (14.29)
  - `Range` (14.35)
- Entity (7)

3. Server Requirements

Your server should be able to understand and properly generate the status codes/methods/headers listed in the previous section. Specifically, your server should:

- Successfully handle HTTP/1.1 and HTTP/1.0 GET method
  - Generate "`404 Not Found`" response when the content could not be found
  - Generate "`200 OK`" when the server can locate and serve the entire request
  - Generate "`206 Partial content`" when the request includes Range header (14.35), or if the requested content cannot be served in a single response
- Make sure that all responses (except status 500) include the Date header. All timestamp in the message must be represented in GMT time zone (RFC 1123-format; 24-hours; abbreviate day of week; month; 4-digit year) For example: `Sun, 06 Nov 1994 08:49:13 GMT`
- You could also interpret/generate additional headers that may help improve the performance (Cache-Control, Age, If-*, X-Content-Duration) However, they are not required.
- Your server should keep the connection alive after receiving a request (unless closed by the client, in which case, the server must also properly close the connection.) If you need to terminate the connection (due to error or capacity problem), you need to include `Connection: close` header in the response.
- For benchmarking / testing purpose, always include `Connection: Keep-Alive` in the response header when the connection remains open after serving a request.
- We will only test your server against a simple binary entity body (identity coding, no entity-header, no chunk encoding – section 4.4 in RFC2616.) This means that the entity length must be equal to the length specified in `Content-Length` response header. Note that the end of the entity body also indicates the end of the HTTP message in a persistent connection.
- Please also make sure that your server generates/recognizes proper end-of-lines with Carriage Return & Line feed characters (`\r\n` – ASCII 13 and 10.) Note the extra `CR-LF` after the request/response header section (just before the entity body).
- For content-type, make sure that your server can support the following media types (based on file extensions or Linux `file` command):

  - `text/plain (.txt)`
  - `text/css (.css)`
  - `text/html (.htm, .html)`
  - `image/gif (.gif)`
  - `image/jpeg (.jpg, .jpeg)`
  - `image/png (.png)`
  - `video/webm (.webm)`
  - `video/mp4 (.mp4)`
  - `application/javascript (.js)`
  - `application/ogg (.ogg)`
  - `application/octet-stream (anything else)`

4. Content Location

By default, the root directory of all content (often known as the content root) is located in the directory "**`content`**" *relative* to the directory from which the server executable file was invoked. For example, if your server is executed from the project directory as followed:

```
/.../project3 $> vodserver #port
```

Then the URL `http://<server-ip-address>:<server-port>/video/video.webm` should correspond to,

```
/.../project3/content/video/video.webm
```

5. Browser Communication

Although implementing a client is not part of this project, you will need to understand the client's behavior in order to successfully implement the server which will allow actual HTML 5 video playback on your favorite browser.

Browser Test: We will use the Firefox 78.8.0esr browser (browser version on ECE cluster machines) to grade the correctness of your server. The process of delivering a file is *roughly* as follows:

- Firefox will initiate the connection when the user makes a request, and it will send an HTTP GET request with the URI of the desired content.
- If the file is "small enough" (what that means is up to you), then the server should respond with a `200 OK` status and should send the entire file contents in the response. This response will conclude this request (although you should still keep the connection alive for future requests).
- If the file is "too big" (once again, up to you), the server should begin streaming the file by responding with a `206 Partial Content` status and sending a chunk of the file. The browser will then periodically make new GET requests that contain Range headers to ask for certain parts of the file, based on the user's needs.
- Make sure that you properly handle the `Last-Modified` response header. Otherwise, browsers will not be able to cache your video.
- Browsers (and different versions of browsers) can all make requests differently, so it will likely take some investigation and maybe some trial and error to figure out the correct way to service your browser's request. In other words, the steps laid out here may *not* be exact and you may need to experiment to figure out what your browser is doing and what it expects.

6. Discussions

I'm confused, where do I start?

First, relax. The first impression of a project is always a bit overwhelming. Make sure you start early. If you are not familiar with network programming, please consult the materials from the lecture and the RFCs. Once you have thought about your server design, you can start by implementing a core server (the echo server is a good start) then implement a message parser module which can parse simple request method (just a simple GET request, ignoring other headers). After you have a basic communication server, you can begin implementing a simple file server that always responds with the entire file. Then, make sure that your server can serve simple files such as html or image files. Once you have verified that your file server works for simple files, you can implement the server range request to handle file streaming. Now, you should be able to test streaming large files (such as an MP4 video of your favorite music video).

How do I measure my server performance?

After you are finished with the functionality, you could focus on the performance part. Since your server is standard-compliant, you could use existing tools such as Apache benchmark; ab (http://httpd.apache.org/docs/2.0/programs/ab.html) to test the scalability of your server. Note that since `ab` only simulates a HTTP/1.0 client, you should include `Connection: Keep-Alive` in the response header to prevent it from hanging. Try to see how many concurrent requests (for a 10KB content) you can have, while still allowing 95% of them to be served within 500ms.

Help! I accidentally delete my working code.

Make a lot of backups! If you do not have the source when the deadline comes, you do not have the project. You may also want to use version control tools to manage your source. The ECE undergraduate cluster machines are equipped with both `git` and `subversion` tools (although learning to use them is not within the scope of this course). Make sure you have a working branch/revision available for final submission before start tuning for performance or adding features. You can always submit an early working version to Canvas.

7. Hand-in Procedure

You will submit your project to Canvas and we will test your code using the browser stated in Section 5. Please create a file named README in the primary submission directory so we know that it is the primary directory that we need to grade (we will ignore the others).

8. Deliverable Items

The deliverables are enumerated as follows,

1. **The source code** for the entire project (if you use external libraries, provide the binaries needed for compilation of your project)
2. **Makefile** - You should prepare a `makefile` for C make (or `build.xml` for Java Ant) which generates the executable file for the project. This should allow us to compile your code from your submission's base directory. For example, the following calls should generate an executable `vodserver` (for C) or `VodServer.class` (for Java), respectively:

```
.../project3$ make
.../project3$ ant
```

3. **Do not submit executable files** (we will DELETE them and deduct your logistic points). However, we must be able to invoke the one of the following commands (after invoking `make/ant`) from your submission directory to start your server on the given port number (e.g. 18441 in this case).

   ```
   $ ./vodserver 18441
   $ java VodServer 18441
   ```

   If you must use a .jar file, please make sure that we can run the following after running ant:

   ```
   $ java -jar VodServer.jar 18441
   ```

4. **A brief design document** regarding your server design in `design.txt / design.pdf` in the submission directory (1-2 pages). Tell us about the following:
   a. Your server design, model and components (add pictures if it helps)
   b. How did you handle multiple clients? How many clients can your server handle effectively?
   c. Libraries used (optional; name, version, homepage-URL; if available)
   d. Extra capabilities you implemented (optional, anything which was not specified in the Server Requirement section)
   e. Extra instructions on how to execute the code, if needed (WARNING: if we cannot compile and run your server on our cluster machine with the above execution method, your logistic points will be deducted. However, please do tell us any runtime parameter your server may have which may help in tuning)

6. Grading

Partial credit will be available according to the following grading scheme,

Items	Points (100 – **441** + 110 – **741**)
Logistics	
- Successful submission & compilation	10
- Design documents	10
Server Correctness	
- Handle standard HTTP GET	10
- Handle HTTP error (404)	10
- Handle HTTP Byte-range request	20
- Play video from browser	10
- Allowing video seek from browser	10
Server Performance	
- < 10% CPU utilization when idle	5
- Handle 10 clients simultaneously	15
Required for 18-741 only (bonus for 18-441)	
- Handle 5000 concurrent clients!	10