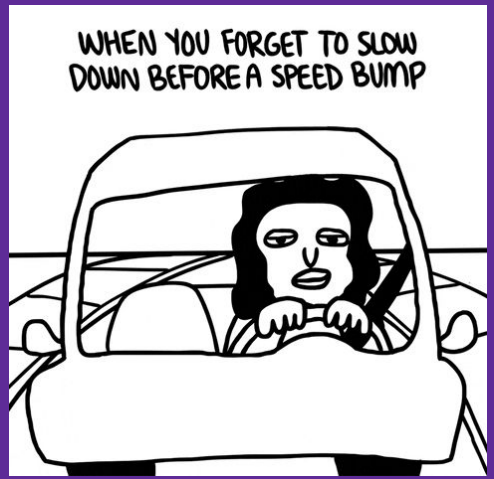
CSE 333 Section 9

HW4, HTTP, and Booooooooooost



Logistics

Friday:

Exercise 11 @ 11 am

Wednesday:

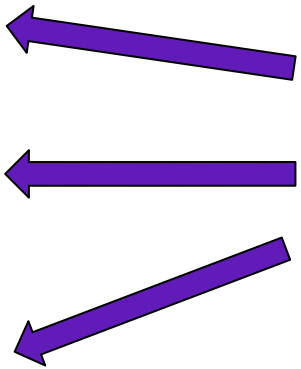
Exercise 12 @ 11 am

Add hw reminder

HW4 Overview

Web Server

1. Establish client connections
    a. Server socket set up
       in hw4/ServerSocket.cc
2. Read client requests
    a. Parse HTTP requests
       in hw4/HttpConnection.cc
3. Respond to requests
    a. Write HTTP responses
       in hw4/HttpServer.cc
4. Fix security vulnerabilities
    a. Escape characters
       in hw4/Utils.cc



Steps 2, 3, and 4 involve a lot of string manipulation which can be tedious!

HTTP Review

HTTP Review

1. What does HTTP stand for?

Hyper**T**ext **T**ransfer **P**rotocol

2. What layer does HTTP reside in?

Application Layer

3. What does HTTP define?

HTTP defines how we should send information between a client and a server

Method URI Version

```
GET /courses/cse333/22wi/ HTTP/1.1
Host: courses.cs.washington.edu
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="98", "Google Chrome";v="98"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,es;q=0.8,it;q=0.7,zh-CN;q=0.6,zh;q=0.5
Cookie: rl_page_init_referrer=RudderEncrypt%3AU2FsdGVkX1%2BPliJ%2Br1vdZYv5Ob9rEBtZr07gXF7fY40%3D; rl_page_init_referring_domain=RudderEncrypt%3AU2FsdGVkX18%2
FC3vkp4W%2BxqaST8KA3F3AquE%2FlamkREM%3D; rl_anonymous_id=RudderEncrypt%3AU2FsdGVkX1%2FMtx35zoGyoUCtalDCjvJFScOb0cibrqiI0NPgclLIZFM8eqsI0L19Lqzn3C86JQTre2ga9Q
rurQ%3D%3D; rl_group_id=RudderEncrypt%3AU2FsdGVkX1%2BSET%2BaLOeiWPUEOBI45OfQyBKKn08Gslw%3D; rl_group_trait=RudderEncrypt%3AU2FsdGVkX1%2Ba%2B0tjYuogrYGTwyCk0p
4F7cmU3X%2ByiqU%3D; rl_user_id=RudderEncrypt%3AU2FsdGVkX1%2BhNfbEzeBvuC9O6SVr2l2oVtvpPpBrna0P2Hn5nsOTKfVCvnFNLiIK; rl_trait=RudderEncrypt%3AU2FsdGVkX19S0AzIw
7sfF830Y8yGyrS8rOttBqA%2FMEY%3D
```

Headers

HTTP Request Format

```
[METHOD] [request-uri] HTTP/[version]\r\n

[headerfield1]: [fieldvalue1]\r\n

[headerfield2]: [fieldvalue2]\r\n

[...]

[headerfieldN]: [fieldvalueN]\r\n

\r\n

[request body, if any]
```

Double return indicates the end of the headers section

HTTP Methods

GET	The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
HEAD	The HEAD method asks for a response identical to that of a GET request, but without the response body.
POST	The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
PUT	The PUT method replaces all current representations of the target resource with the request payload.
DELETE	The DELETE method deletes the specified resource.
CONNECT	The CONNECT method establishes a tunnel to the server identified by the target resource.
OPTIONS	The OPTIONS method is used to describe the communication options for the target resource.
TRACE	The TRACE method performs a message loop-back test along the path to the target resource.
PATCH	The PATCH method is used to apply partial modifications to a resource.

Version

```
HTTP/1.1 200 OK
Date: Mon, 21 May 2018 07:58:46 GMT
Server: Apache/2.2.32 (Unix) mod_ssl/2.2.32 OpenSSL/1.0.1e-fips
mod_pubcookie/3.3.4a mod_uwa/3.2.1 Phusion_Passenger/3.0.11
Last-Modified: Mon, 21 May 2018 07:58:05 GMT
ETag: "2299e1ef-52-56cb2a9615625"
Accept-Ranges: bytes
Content-Length: 82
Vary: Accept-Encoding,User-Agent
Connection: close
Content-Type: text/html
Set-Cookie:
bbbbbbbbbbbbbbb=DBMLFDMJCGAOILMBPIIAAIFLGBAKOJNNMCJIKKBKCDMDEJHMPONHCILPIBL
ADEAKCIABMEEPAOPMMKAOLHOKJMIGMIDKIHNCANAPHMFMBLBABPFENPDANJAPIBOIOOOD;
HttpOnly

<html><body>
<font color="chartreuse" size="18pt">Awesome!!</font>
</body></html>
```

Status

Headers

Request body

HTTP Response Format

```
HTTP/[version] [status code] [reason]\r\n
[headerfield1]: [fieldvalue1]\r\n
[headerfield2]: [fieldvalue2]\r\n
[...]
[headerfieldN]: [fieldvalueN]\r\n
\r\n
[response body, if any]
```

Double return indicates the end of the headers section

HTTP Response Status Codes

- `HTTP/1.1 200 OK`
  - The request succeeded and the requested object is sent
- `HTTP/1.1 404 Not Found`
  - The requested object was not found
- `HTTP/1.1 301 Moved Permanently`
  - The object exists, but its name has changed
  - The new URL is given as the "`Location:`" header value
- `HTTP/1.1 500 Server Error`
  - The server had some kind of unexpected error

HTTP REQUEST DEMO (telnet)

Using Telnet with HW4

1. Launch the server

   ```
   ./http333d <port> ../projdocs/ unit_test_indices/*
   ```

2. Connect with telnet

   ```
   telnet <HostName> <port>
   ```

3. Write an HTTP request and send it

4. To exit telnet:
   - **Ctrl+]** then **Ctrl+d**

Writing an HTTP Request

- Example HTTP Request layout can be found in `HttpRequest.h`
- Example file request:
  - `GET /static/test_tree/books/artofwar.txt HTTP/1.1`
- Example query request:
  - `GET /query?terms=books+of+war HTTP/1.1`
- To send a request, hit [Enter] *twice*
- Compare the output of `solution_binaries/http333d` to `./http333d`

BOOOOOOOOST

BOOST

Boost is a free C++ library that provides support for various tasks in C++

- **Note:** Boost does NOT follow the Google style guide!!!

Boost adds many string algorithms that you may have seen in Java

- Include with `#include <boost/algorithm/string.hpp>`

We are showcasing a few we think could be useful for HW4, but more can be found here:

- https://www.boost.org/doc/libs/1_60_0/doc/html/string_algo.html

trim

```
void boost::trim(string& input);
```

- Removes all leading and trailing whitespace from the string
- `input` is an input *and* output parameter (non-const reference)

```
string s("    HI    ");
boost::algorithm::trim(s);

// results in s == "HI"
```

replace_all

```
void boost::replace_all(string& input, const string& search,
                                       const string& format);
```

- Replaces all instances of `search` inside `input` with `format`

```
string s("ynrnrt");
boost::algorithm::replace_all(s, "nr", "e");

// results in s == "yeet"
```

replace_all

```
void boost::replace_all(string& input, const string& search,
                                       const string& format);
```

- Replaces all instances of `search` inside `input` with `format`

```
string s("queue?");
boost::algorithm::replace_all(s, "que", "q");
// results in s == "que?"
```

`replace_all()` guarantees that '`format`' will be in the final result if-and-only-if '`search`' existed.

`replace_all()` makes a *single* pass over `input`.

split

```
void boost::split(vector<string>& output,
          const string& input,
          boost::PredicateT match_on,
          boost::token_compress_mode_type compress);
```

- Split the string by the characters in `match_on`

```
boost::PredicateT boost::is_any_of(const string& tokens);
```

- Returns predicate that matches on any of the characters in tokens

split Examples

```
vector<string> tokens;
string s("I-am--split");

boost::split(tokens, s, boost::is_any_of("-"),
                        boost::token_compress_on);
// results in tokens == ["I", "am", "split"]

boost::split(tokens, s, boost::is_any_of("-"),
                        boost::token_compress_off);
// results in tokens == ["I", "am", "", “split"]
```

Exercise 1

Write a function called `ExtractRequestLine` that takes in a well-formatted HTTP request as a `string` and returns a `map` with the keys as `method`, `uri`, `version` and the values from the corresponding request. For example,

Example Input:

```
"GET /index.html HTTP/1.1\r\nHost: www.mywebsite.com\r\nConnect
ion: keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n"
```

Map Returned:

```
{
    "method"  : "GET"
    "uri"     : "/index.html"
    "version" : "HTTP/1.1"
}
```

Exercise 1

```
map<string,string> ExtractRequestLine(const string& request) {
  vector<string> lines;
  boost::split(lines, request, boost::is_any_of("\r\n"),
               boost::token_compress_on);
  vector<string> components;
  string firstLine = lines[0];
  boost::split(components, firstLine, boost::is_any_of(" "),
               boost::token_compress_on);
  map<string, string> res;
  res["method"] = components[0];
  res["uri"] = components[1];
  res["version"] = components[2];
  return res;
}
```

Exercise 2

Write a function that takes in a string that contains words separated by whitespace and returns a vector that contains all of the words in that string, in the same order as they show up, but with no duplicates. Ignore all leading and trailing whitespace in the input string.

Example:
`RemoveDuplicates(" Hi I'm sorry  jon sorry hi hihi  hi hi     ")`
should return the vector `["Hi", "I'm", "sorry", "jon", "hi", "hihi"]`

```
vector<string> RemoveDuplicates(const string& input){
   string copy(input);
   boost::algorithm::trim(copy);
   std::vector<string> components;
   boost::split(components, copy, boost::is_any_of(" \t\n"),
                boost::token_compress_on);

   std::vector<string> result;
   std::set<string> unique_components;
   for (const auto& comp : components) {
     if (unique_components.find(comp) == aux.end()) {
        result.push_back(comp);
        unique_components.insert(comp);

      }

   }

   return result;

}
```