CSE 333

Lecture 19 -- HTTP

Hal Perkins

Department of Computer Science & Engineering

University of Washington

Administrivia

HW4 due last Thursday of the quarter (a week from tomorrow!!!!)

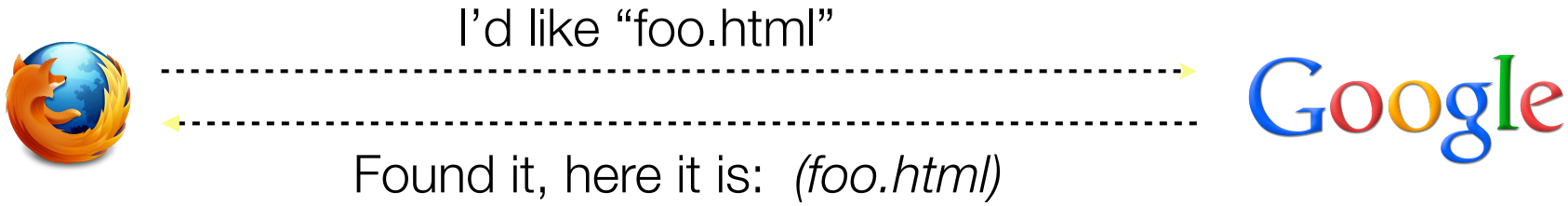
- How's it look?

New exercise covering server-side programming posted now, due Friday before class (to get it out of the way before the weekend)

Which means we could fit in one more exercise on threads, due Monday before class. Thoughts?

Sections tomorrow: threads & concurrency (first look)

Let's dive down into HTTP



A client establishes one or more TCP connections to a server

- the client sends a request for a web object over a connection, and the server replies with the object's contents
- we have to figure out how let the client and server communicate their intentions to each other clearly
- we have to define a ***protocol***

HTTP is a "protocol"

Protocol: the rules governing the exchange of messages, and the format of those messages, in a computing system

- what messages can a client exchange with a server?
  - what do the messages mean?
  - what are legal replies to a message?
  - what is the syntax of a message?
- what sequence of messages is legal?
  - how are errors conveyed?

A protocol is (roughly) the network equivalent of an API

HTTP

Hyper**t**ext **t**ransport **p**rotocol

- a request / response protocol
  - a client (web browser) sends a request to a web server
  - the server processes the request, sends a response
- typically, a request asks the server to retrieve a resource
  - a resource is an object or document, named by a URI
- a response indicates whether the server succeeded
  - and, if so, it provides the content of the requested response

An HTTP request

```
[METHOD] [request-uri] HTTP/[version]\r\n
[fieldname1]: [fieldvalue1]\r\n
[fieldname2]: [fieldvalue2]\r\n
[...]
[fieldnameN]: [fieldvalueN]\r\n
\r\n
[request body, if any]
```

let's use "nc" to see a real request

HTTP methods

There are three commonly used HTTP methods

- **GET**: "please send me the named document"
- **POST**: "I'd like to submit data to you, such as a file upload"
- **HEAD**: "send me the headers for the named object, but not the object. (I'd like to see if my cached copy is still valid.)"

There are several rarely used methods:

- PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH, ...
  - TRACE: "if there are any proxies or caches in between me and the server, please speak up!"

HTTP versions

Most browsers and servers speak HTTP/1.1

- "version 1.1 of the HTTP protocol"
  - http://www.w3.org/Protocols/rfc2616/rfc2616.html
- introduced around 1996 to fix shortcomings of HTTP/1.0
  - better performance, richer caching features, better support for multi-homed servers, and much more
  - more complicated to implement than HTTP/1.0

Client headers

The client can provide zero or more request "headers"

- they provide information to the server, or modify how the server should process the request

You'll encounter many in practice

- Host: the DNS name of the server [why?]
- User-Agent: an identifying string naming the browser [why?]
- Accept: the content types the client prefers or can accept
- Cookie: an HTTP cookie previously set by the server

Example...

```
GET /foo/bar.html HTTP/1.1
Host: futureproof.cs.washington.edu:5555
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2)
AppleWebKit/536.26.17 (KHTML, like Gecko) Version/6.0.2 Safari/
536.26.17
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: __utma=59807807.1547453334.1214335349.1301330421.1301339949.
30; __utmz=59807807.1300728257.27.14.utmcsr=google|utmccn=(organic)|
utmcmd=organic|utmctr=csgordon@u.washington.edu;
__utma=80390417.1521666831.1201286098.1302710464.1302717901.34;
__utmz=80390417.1301950604.31.15.utmcsr=cs.washington.edu|utmccn=
(referral)|utmcmd=referral|utmcct=/education/courses/cse333/11sp/;
__qca=P0-1872143622-1294952393928
Connection: keep-alive
```

An HTTP response

```
HTTP/[version] [status code] [reason]\r\n
[fieldname1]: [fieldvalue1]\r\n
[fieldname2]: [fieldvalue2]\r\n
[...]
[fieldnameN]: [fieldvalueN]\r\n
\r\n
[response body, if any]
```

let's use "telnet" to see a real response

Status codes, reason phrase

Code: a computer-readable outcome of the request

- three digit integer; first digit identifies the response category
  - 1xx: some kind of informational message
  - 2xx: success of some kind
  - 3xx: redirects the client to a different URL
  - 4xx: the client's request contained some error
  - 5xx: the server experienced an error

Reason phrase: human-readable explanation

- e.g., "OK" or "Moved Temporarily"

Common status lines

HTTP/1.1 200 OK

- the request succeeded, 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 in the “Location:” header

HTTP/1.1 500 Server Error

- the server had some kind of unexpected error

Server headers

The server can provide zero or more response "headers"

- they provide information to the client, or modify how the client should process the response

You'll encounter many in practice

- **Server**: a string identifying the server software [why?]
- **Content-Type:** the type of the requested object
- **Content-Length:** size of requested object [why?]
- **Last-Modified:** a date indicating the last time the request object was modified [why?]

Example

```
HTTP/1.1 200 OK
Date: Fri, 27 May 2011 17:05:53 GMT
Server: Apache/2.2.19 (Fedora)
Last-Modified: Fri, 27 May 2011 17:04:51 GMT
ETag: "2740640-52-4a444ef9392c0"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Content-Language: en

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

Cool HTTP/1.1 features

"Chunked Transfer-Encoding"

- a server might not know how big a response object is
  - e.g., you're dynamically generating the content in response to a query or other user input
- how do you send Content-Length?
  - could wait until you've finished generating the response, but that's not great in terms of **latency**
  - instead, want to start sending response right away
- chunked message body: response is series of chunks
  - try with http://www.cs.washington.edu/

Cool HTTP/1.1 features

Persistent connections

- establishing a TCP connection is costly
  - multiple network "round trips" just to set up the TCP connection
  - TCP has a feature called "slow start"; slowly grows the rate at which a TCP connection transmits to avoid overwhelming networks
- a web page consists of multiple objects, and a client probably visits several pages on the same server
  - bad idea: separate TCP connection for each object
  - better idea: single TCP connection, multiple requests
  - *try it on www.cs.washington.edu*

See you on Friday!