Web Services

15-213 / 18-213: Introduction to Computer Systems
22nd Lecture, Nov. 12, 2013

Instructors:
Randy Bryant, Dave O’Hallaron, and Greg Kesden
Web History

- **1989:**
  - Tim Berners-Lee (CERN) writes internal proposal to develop a distributed hypertext system
    - Connects “a web of notes with links”
    - Intended to help CERN physicists in large projects share and manage information

- **1990:**
  - Tim BL writes a graphical browser for Next machines
Web History (cont)

■ 1992
  ▪ NCSA server released
  ▪ 26 WWW servers worldwide

■ 1993
  ▪ Marc Andreessen releases first version of NCSA Mosaic browser
  ▪ Mosaic version released for (Windows, Mac, Unix)
  ▪ Web (port 80) traffic at 1% of NSFNET backbone traffic
  ▪ Over 200 WWW servers worldwide

■ 1994
  ▪ Andreessen and colleagues leave NCSA to form “Mosaic Communications Corp” (predecessor to Netscape)
Web Servers

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (eventually)

- Current version is HTTP/1.1
  - RFC 2616, June, 1999.

http://www.w3.org/Protocols/rfc2616/rfc2616.html
Web Content

- Web servers return *content* to clients
  - *content*: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type

- Example MIME types
  - `text/html`: HTML document
  - `text/plain`: Unformatted text
  - `application/postscript`: Postscript document
  - `image/gif`: Binary image encoded in GIF format
  - `image/jpeg`: Binary image encoded in JPEG format
Static and Dynamic Content

- The content returned in HTTP responses can be either static or dynamic
  - **Static content**: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
    - Request identifies which content file
  - **Dynamic content**: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
    - Request identifies which file containing executable code

- **Bottom line**: *(most) Web content is associated with a file that is managed by the server*
URLs and how clients and servers use them

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: `http://www.cmu.edu:80/index.html`
- Clients use **prefix** (`http://www.cmu.edu:80`) to infer:
  - What kind (protocol) of server to contact (HTTP)
  - Where the server is (`www.cmu.edu`)
  - What port it is listening on (80)
- Servers use **suffix** (`/index.html`) to:
  - Determine if request is for static or dynamic content.
    - No hard and fast rules for this
    - Convention: executables reside in `cgi-bin` directory
  - Find file on file system
    - Initial “/” in suffix denotes home directory for requested content.
    - Minimal suffix is “/”, which server expands to configured default filename (usually, `index.html`)
Example of an HTTP Transaction

```
unix> telnet www.cmu.edu 80
Trying 128.2.10.162...
Connected to www.cmu.edu.
Escape character is '^[].
GET / HTTP/1.1
host: www.cmu.edu

HTTP/1.1 301 Moved Permanently
Location: http://www.cmu.edu/index.shtml

Connection closed by foreign host.
unix>`
Example of an HTTP Transaction, Take 2

unix> telnet www.cmu.edu 80
Trying 128.2.10.162...
Connected to www.cmu.edu.
Escape character is '^]'.
GET /index.shtml HTTP/1.1
host: www.cmu.edu

HTTP/1.1 200 OK
Date: Fri, 29 Oct 2010 19:41:08 GMT
Server: Apache/1.3.39 (Unix) mod_pubcookie/3.3.3 ...
Transfer-Encoding: chunked
Content-Type: text/html

Lots of stuff

Connection closed by foreign host.
unix>
HTTP Requests

- HTTP request is a request line, followed by zero or more request headers

Request line: `<method> <uri> <version>`
- `<method>` is one of GET, POST, OPTIONS, HEAD, PUT, DELETE, or TRACE
- `<uri>` is typically URL for proxies, URL suffix for servers
  - A URL is a type of URI (Uniform Resource Identifier)
  - See http://www.ietf.org/rfc/rfc2396.txt
- `<version>` is HTTP version of request (HTTP/1.0 or HTTP/1.1)
HTTP Requests (cont)

- **HTTP methods:**
  - **GET**: Retrieve static or dynamic content
    - Arguments for dynamic content are in URI
    - Workhorse method (99% of requests)
  - **POST**: Retrieve dynamic content
    - Arguments for dynamic content are in the request body
  - **OPTIONS**: Get server or file attributes
  - **HEAD**: Like GET but no data in response body
  - **PUT**: Write a file to the server!
  - **DELETE**: Delete a file on the server!
  - **TRACE**: Echo request in response body
    - Useful for debugging

- **Request headers**: `<header name>`: `<header data>`
  - Provide additional information to the server
HTTP Versions

- Major differences between HTTP/1.1 and HTTP/1.0
  - HTTP/1.0 uses a new connection for each transaction
  - HTTP/1.1 also supports *persistent connections*
    - multiple transactions over the same connection
    - Connection: Keep-Alive
  - HTTP/1.1 requires **HOST** header
    - Host: [www.cmu.edu](http://www.cmu.edu)
    - Makes it possible to host multiple websites at single Internet host
  - HTTP/1.1 supports *chunked encoding* (described later)
    - Transfer-Encoding: chunked
  - HTTP/1.1 adds additional support for caching
HTTP Responses

HTTP response is a **response line** followed by zero or more **response headers**, possibly followed by data

**Response line:**

```
<version> <status code> <status msg>
```

- `<version>` is HTTP version of the response
- `<status code>` is numeric status
- `<status msg>` is corresponding English text
  - 200 OK Request was handled without error
  - 301 Moved Provide alternate URL
  - 403 Forbidden Server lacks permission to access file
  - 404 Not found Server couldn’t find the file

**Response headers:** `<header name>`: `<header data>`

- Provide additional information about response
- **Content-Type**: MIME type of content in response body
- **Content-Length**: Length of content in response body
GET Request to Apache Server
From Firefox Browser

URI is just the suffix, not the entire URL

```
GET /~bryant/test.html HTTP/1.1
Host: www.cs.cmu.edu
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv: 1.9.2.11) Gecko/20101012 Firefox/3.6.11
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
CRLF (\r\n)
```
GET Response From Apache Server

HTTP/1.1 200 OK
Date: Fri, 29 Oct 2010 19:48:32 GMT
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.7m mod_pubcookie/3.3.2b PHP/5.3.1
Accept-Ranges: bytes
Content-Length: 479
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
<html>
<head><title>Some Tests</title></head>

<body>
<h1>Some Tests</h1>
...
</body>
</html>
Proxies

- A **proxy** is an intermediary between a client and an origin server
  - To the client, the proxy acts like a server
  - To the server, the proxy acts like a client

```
Client          Proxy          Origin Server
1. Client request
               2. Proxy request
               3. Server response
               4. Proxy response
```
Why Proxies?

- Can perform useful functions as requests and responses pass by
  - Examples: Caching, logging, anonymization, filtering, transcoding
Tiny Web Server

- Tiny Web server described in text
  - Tiny is a sequential Web server
  - Serves static and dynamic content to real browsers
    - text files, HTML files, GIF and JPEG images
  - 226 lines of commented C code
  - Not as complete or robust as a real web server
Tiny Operation

- Accept connection from client
- Read request from client (via connected socket)
- Split into method / uri / version
  - If not GET, then return error
- If URI contains “cgi-bin” then serve dynamic content
  - (Would do wrong thing if had file “abcgi-bingo.html”)
  - Fork process to execute program
- Otherwise serve static content
  - Copy file to output
Tiny Serving Static Content

/* Send response headers to client */
get_filetype(filename, filetype);
sprintf(buf, "HTTP/1.0 200 OK\r\n");
sprintf(buf, "Server: Tiny Web Server\r\n", buf);
sprintf(buf, "Content-length: %d\r\n", bufsize);
sprintf(buf, "Content-type: %s\r\n\n", buf, filetype);
Rio_writen(fd, buf, strlen(buf));

/* Send response body to client */
srcfd = Open(filename, O_RDONLY, 0);
srcp = Mmap(0, bufsize, PROT_READ, MAP_PRIVATE, srcfd, 0);
Close(srcfd);
Rio_writen(fd, srcp, bufsize);
Munmap(srcp, bufsize);

- Serve file specified by filename
- Use file metadata to compose header
- "Read" file via mmap
- Write to output
Serving Dynamic Content

- Client sends request to server
- If request URI contains the string "/cgi-bin", then the server assumes that the request is for dynamic content
Serving Dynamic Content (cont)

- The server creates a child process and runs the program identified by the URI in that process
Serving Dynamic Content (cont)

- The child runs and generates the dynamic content
- The server captures the content of the child and forwards it without modification to the client
Issues in Serving Dynamic Content

- How does the client pass program arguments to the server?
- How does the server pass these arguments to the child?
- How does the server pass other info relevant to the request to the child?
- How does the server capture the content produced by the child?
- These issues are addressed by the Common Gateway Interface (CGI) specification.
CGI

- Because the children are written according to the CGI spec, they are often called *CGI programs* or *CGI scripts*.

- However, CGI really defines a simple standard for transferring information between the client (browser), the server, and the child process.

- CGI is the original standard for generating dynamic content. Has been largely replaced by other, faster techniques:
  - E.g., fastCGI, Apache modules, Java servlets
  - Avoid having to create process on the fly.
The add.com Experience

input URL  host  port  CGI program  args

Welcome to add.com: THE Internet addition portal.

The answer is: 15213 + 18243 -> 33456

Thanks for visiting!
Serving Dynamic Content With GET

- **Question**: How does the client pass arguments to the server?
- **Answer**: The arguments are appended to the URI.

Can be encoded directly in a URL typed to a browser or a URL in an HTML link:

- `http://add.com/cgi-bin/adder?n1=15213&n2=18243`
- `adder` is the CGI program on the server that will do the addition.
- Argument list starts with “?”
- Arguments separated by “&”
- Spaces represented by “+” or “%20”
Serving Dynamic Content With GET

- **URL:**
  - `cgi-bin/adder?n1=15213&n2=18243`

- **Result displayed on browser:**

```
Welcome to add.com: THE Internet addition portal. The answer is: 15213 + 18243 -> 33456
Thanks for visiting!
```
Serving Dynamic Content With GET

- **Question:** How does the server pass these arguments to the child?

- **Answer:** In environment variable **QUERY_STRING**
  - A single string containing everything after the “?”
  - For add: `QUERY_STRING = "n1=15213&n2=18243"

```c
if ((buf = getenv("QUERY_STRING")) != NULL) {
    if (sscanf(buf, "%d=%d&%d=%d\n", &n1, &n2) == 2)
        sprintf(msg, "%d + %d -> %d\n", n1, n2, n1+n2);
    else
        sprintf(msg, "Can't parse buffer '%s'\n", buf);
}
```

From `adder.c`
Additional CGI Environment Variables

- **General**
  - `SERVER_SOFTWARE`
  - `SERVER_NAME`
  - `GATEWAY_INTERFACE` (CGI version)

- **Request-specific**
  - `SERVER_PORT`
  - `REQUEST_METHOD` (GET, POST, etc)
  - `QUERY_STRING` (contains GET args)
  - `REMOTE_HOST` (domain name of client)
  - `REMOTE_ADDR` (IP address of client)
  - `CONTENT_TYPE` (for POST, type of data in message body, e.g., text/html)
  - `CONTENT_LENGTH` (length in bytes)
Even More CGI Environment Variables

- In addition, the value of each header of type *type* received from the client is placed in environment variable `HTTP_type`
  - Examples (any “-” is changed to “_”):
    - `HTTP_ACCEPT`
    - `HTTP_HOST`
    - `HTTP_USER_AGENT`
Serving Dynamic Content With GET

- **Question:** How does the server capture the content produced by the child?
- **Answer:** The child generates its output on `stdout`. Server uses `dup2` to redirect `stdout` to its connected socket.
  - Notice that only the child knows the type and size of the content. Thus the child (not the server) must generate the corresponding headers.

```c
/* Make the response body */
sprintf(content, "Welcome to add.com: ");
sprintf(content, "%sTHE Internet addition portal.\r\n<p>", content);
sprintf(content, "%sThe answer is: %s\r\n<p>", content, msg);
sprintf(content, "%sThanks for visiting!\r\n", content);

/* Generate the HTTP response */
printf("Content-length: %u\r\n", (unsigned) strlen(content));
printf("Content-type: text/html\r\n\r\n"),
printf("%s", content);
```

From `adder.c`
Serving Dynamic Content With GET

```
linux> telnet greatwhite.ics.cs.cmu.edu 15213
Trying 128.2.220.10...
Connected to greatwhite.ics.cs.cmu.edu (128.2.220.10).
Escape character is '^]'.
GET /cgi-bin/adder?n1=5&n2=27 HTTP/1.1
host: greatwhite.ics.cs.cmu.edu

HTTP request sent by client

HTTP/1.0 200 OK
Server: Tiny Web Server
Content-length: 109
Content-type: text/html

Welcome to add.com: THE Internet addition portal.
<p>The answer is: 5 + 27 -> 32

HTTP response generated by the server

HTTP response generated by the CGI program

<p>Thanks for visiting!
Connection closed by foreign host.
```
Tiny Serving Dynamic Content

/* Return first part of HTTP response */
sprintf(buf, "HTTP/1.0 200 OK\r\n");
Rio_writen(fd, buf, strlen(buf));
sprintf(buf, "Server: Tiny Web Server\r\n");
Rio_writen(fd, buf, strlen(buf));

if (Fork() == 0) { /* child */
    /* Real server would set all CGI vars here */
    setenv("QUERY_STRING", cgiargs, 1);
    Dup2(fd, STDOUT_FILENO); /* Redirect stdout to client */
    Execve(filename, emptylist, environ); /* Run CGI prog */
}
Wait(NULL); /* Parent waits for and reaps child */

- Fork child to execute CGI program
- Change stdout to be connection to client
- Execute CGI program with execve
Data Transfer Mechanisms

- **Standard**
  - Specify total length with content-length
  - Requires that program buffer entire message

- **Chunked**
  - Break into blocks
  - Prefix each block with number of bytes (Hex coded)
Chunked Encoding Example

HTTP/1.1 200 OK
Date: Sun, 31 Oct 2010 20:47:48 GMT
Server: Apache/1.3.41 (Unix)
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<table>
<thead>
<tr>
<th>First Chunk: 0xd75 = 3445 bytes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Second Chunk: 0 bytes (indicates last chunk)</th>
</tr>
</thead>
</table>
For More Information

- Study the Tiny Web server described in your text
  - Tiny is a sequential Web server.
  - Serves static and dynamic content to real browsers.
    - text files, HTML files, GIF and JPEG images.
  - 220 lines of commented C code.
  - Also comes with an implementation of the CGI script for the add.com addition portal.

- See the HTTP/1.1 standard:
  - [http://www.w3.org/Protocols/rfc2616/rfc2616.html](http://www.w3.org/Protocols/rfc2616/rfc2616.html)