

**CSCE 230J**  
**Computer Organization**

## **Web Services**

**Dr. Steve Goddard**  
***goddard@cse.unl.edu***

*<http://cse.unl.edu/~goddard/Courses/CSCE230J>*

## **Giving credit where credit is due**

- **Most of slides for this lecture are based on slides created by Drs. Bryant and O'Hallaron, Carnegie Mellon University.**
- **I have modified them and added new slides.**

## Topics

- HTTP
- Serving static content
- Serving dynamic content

3

## Web History

**1945:**

- Vannevar Bush, “As we may think”, Atlantic Monthly, July, 1945.
  - Describes the idea of a distributed hypertext system.
  - A “memex” that mimics the “web of trails” in our minds.

**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.

4

## 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.

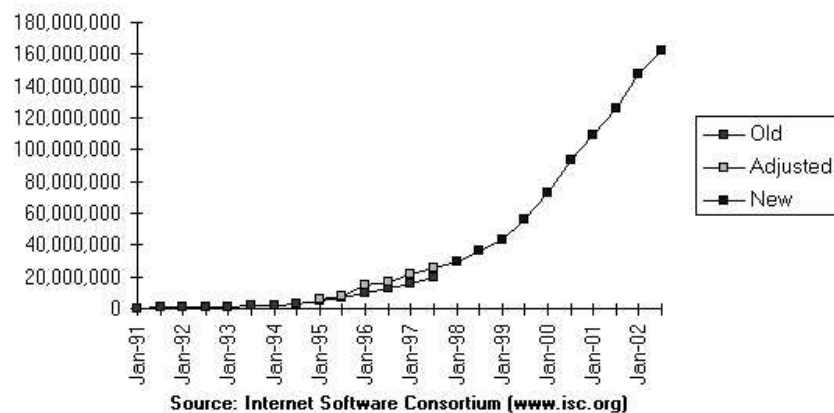
### 1994

- Andreessen and colleagues leave NCSA to form "Mosaic Communications Corp" (now Netscape).

5

## Internet Hosts

Internet Domain Survey Host Count



6

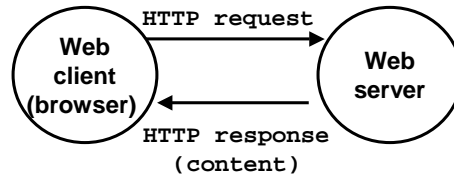
# Web Servers

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 (usually)

Current version is HTTP/1.1

- RFC 2616, June, 1999.



7

# 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 |

8

# 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.
- **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.

**Bottom line:** *All Web content is associated with a file that is managed by the server.*

9

# URLs

Each file managed by a server has a unique name called a URL (Universal Resource Locator)

**URLs for static content:**

- `http://www.cs.cmu.edu:80/index.html`
- `http://www.cs.cmu.edu/index.html`
- `http://www.cs.cmu.edu`
  - Identifies a file called `index.html`, managed by a Web server at `www.cs.cmu.edu` that is listening on port 80.

**URLs for dynamic content:**

- `http://www.cs.cmu.edu:8000/cgi-bin/adder?15000&213`
  - Identifies an executable file called `adder`, managed by a Web server at `www.cs.cmu.edu` that is listening on port 8000, that should be called with two argument strings: 15000 and 213.

10

## How Clients and Servers Use URLs

Example URL: `http://www.aol.com:80/index.html`

Clients use *prefix* (`http://www.aol.com:80`) to infer:

- What kind of server to contact (Web server)
- Where the server is (`www.aol.com`)
- 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 all servers expand to some default home page (e.g., `index.html`).

11

## Anatomy of an HTTP Transaction

```
unix> telnet www.aol.com 80
Trying 205.188.146.23...
Connected to aol.com.
Escape character is '^]'.
GET / HTTP/1.1
host: www.aol.com
```

```
HTTP/1.0 200 OK
MIME-Version: 1.0
Date: Mon, 08 Jan 2001 04:59:42 GMT
Server: NaviServer/2.0 AOLserver/2.3.3
Content-Type: text/html
Content-Length: 42092
```

```
<html>
...
</html>
Connection closed by foreign host.
unix>
```

*Client: open connection to server  
Telnet prints 3 lines to the terminal*

*Client: request line  
Client: required HTTP/1.1 HOST header  
Client: empty line terminates headers.  
Server: response line  
Server: followed by five response headers*

*Server: expect HTML in the response body  
Server: expect 42,092 bytes in the resp body  
Server: empty line ("`\r\n`") terminates hdrs  
Server: first HTML line in response body  
Server: 766 lines of HTML not shown.  
Server: last HTML line in response body  
Server: closes connection  
Client: closes connection and terminates*

12

# HTTP Requests

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

**Request line:** <method> <uri> <version>

- <version> is HTTP version of request (HTTP/1.0 or HTTP/1.1)
- <uri> is typically URL for proxies, URL suffix for servers.
- <method> is either GET, POST, OPTIONS, HEAD, PUT, DELETE, or TRACE.

13

## 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.

14

## HTTP Requests (cont)

Request headers: `<header name>: <header data>`

- Provide additional information to the server.

### 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: kittyhawk.cmcl.cs.cmu.edu
- HTTP/1.1 adds additional support for caching

15

## HTTP Responses

HTTP response is a *response line* followed by zero or more *response headers*.

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
  - 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.

16



## GET Request to Apache Server From IE Browser

```
GET /test.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
Host: euro.ecom.cmu.edu
Connection: Keep-Alive
CRLF (\r\n)
```

17

## GET Response From Apache Server

```
HTTP/1.1 200 OK
Date: Thu, 22 Jul 1999 04:02:15 GMT
Server: Apache/1.3.3 Ben-SSL/1.28 (Unix)
Last-Modified: Thu, 22 Jul 1999 03:33:21 GMT
ETag: "48bb2-4f-37969101"
Accept-Ranges: bytes
Content-Length: 79
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
CRLF
<html>
<head><title>Test page</title></head>
<body>
<h1>Test page</h1>
</html>
```

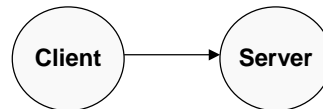
18

## 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.**

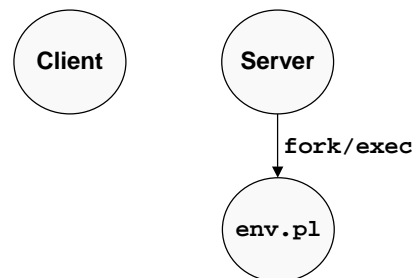
GET /cgi-bin/env.pl HTTP/1.1



19

## Serving Dynamic Content (cont)

**The server creates a child process and runs the program identified by the URI in that process**

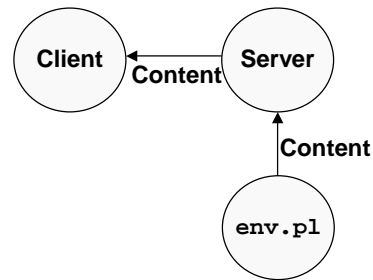


20

## 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



21

## Issues in Serving Dynamic Content

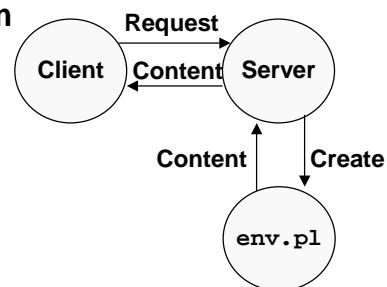
How does the client pass program arguments to the server?

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.



22

## **CGI**

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

**Because many CGI programs are written in Perl, they are often called *CGI scripts*.**

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

23

## **add.com: THE Internet addition portal!**

**Ever need to add two numbers together and you just can't find your calculator?**

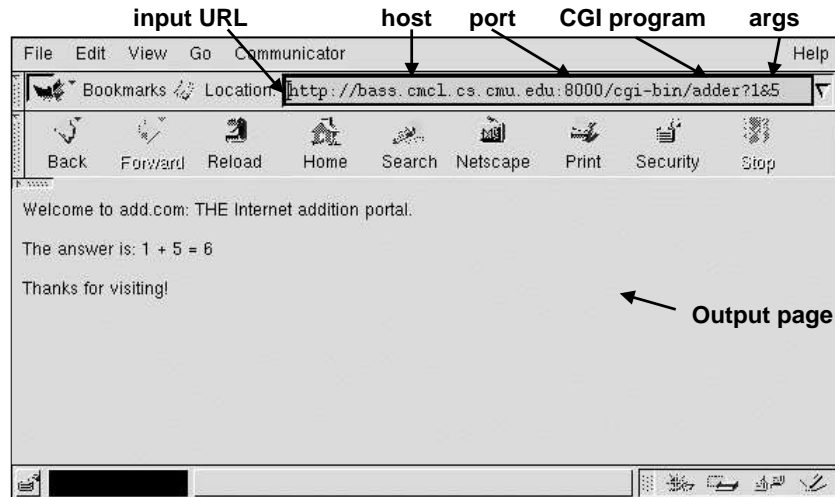
**Try Dr. Dave's addition service at "add.com: THE Internet addition portal!"**

- Takes as input the two numbers you want to add together.
- Returns their sum in a tasteful personalized message.

**After the IPO we'll expand to multiplication!**

24

## The add.com Experience



25

## 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?1&2`
- `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"

Can also be generated by an HTML form

```
<form method=get action="http://add.com/cgi-bin/postadder">
```

26

## Serving Dynamic Content With GET

### URL:

- `http://add.com/cgi-bin/adder?1&2`

### Result displayed on browser:

Welcome to add.com: THE Internet addition portal.

The answer is:  $1 + 2 = 3$

Thanks for visiting! Tell your friends.

27

## 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.com`: `QUERY_STRING = “1&2”`

```
/* child code that accesses the argument list */
if ((buf = getenv("QUERY_STRING")) == NULL) {
    exit(1);
}

/* extract arg1 and arg2 from buf and convert */
...
n1 = atoi(arg1);
n2 = atoi(arg2);
```

28

## Serving Dynamic Content With GET

**Question:** How does the server pass other info relevant to the request to the child?

**Answer:** In a collection of environment variables defined by the CGI spec.

29

## Some 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)

30

## Some 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:

- `HTTP_ACCEPT`
- `HTTP_HOST`
- `HTTP_USER_AGENT` (any “-” is changed to “\_”)

31

## 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.

```
/* child generates the result string */
sprintf(content, "Welcome to add.com: THE Internet addition portal\
<p>The answer is: %d + %d = %d\
<p>Thanks for visiting!\r\n",
        n1, n2, n1+n2);

/* child generates the headers and dynamic content */
printf("Content-length: %d\r\n", strlen(content));
printf("Content-type: text/html\r\n");
printf("\r\n");
printf("%s", content);
```

32



## Serving Dynamic Content With GET

```
bass> ./tiny 8000
GET /cgi-bin/adder?1&2 HTTP/1.1
Host: bass.cmcl.cs.cmu.edu:8000
<CRLF>
```

-----

```
kittyhawk> telnet bass 8000
Trying 128.2.222.85...
Connected to BASS.CMCL.CS.CMU.EDU.
Escape character is '^]'.
GET /cgi-bin/adder?1&2 HTTP/1.1
Host: bass.cmcl.cs.cmu.edu:8000
<CRLF>
```

-----

```
HTTP/1.1 200 OK
Server: Tiny Web Server
Content-length: 102
Content-type: text/html
<CRLF>
Welcome to add.com: THE Internet addition portal.
<p>The answer is: 1 + 2 = 3
<p>Thanks for visiting!
<CRLF>
```

-----

```
Connection closed by foreign host.
kittyhawk>
```

HTTP request received by  
Tiny Web server

HTTP request sent by client

HTTP response generated by  
the server

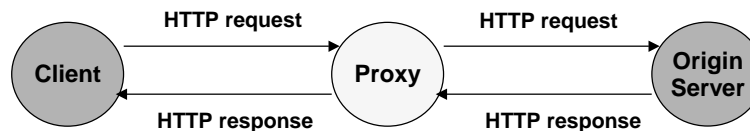
HTTP response generated by  
the CGI program

33

## 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.

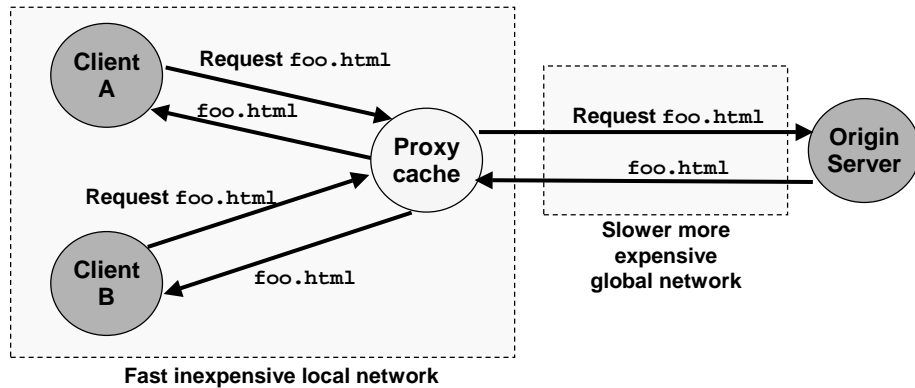


34

## Why Proxies?

Can perform useful functions as requests and responses pass by

- Examples: Caching, logging, anonymization



35

## 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.

36