

15-213

Sockets and Web services

April 23, 2002

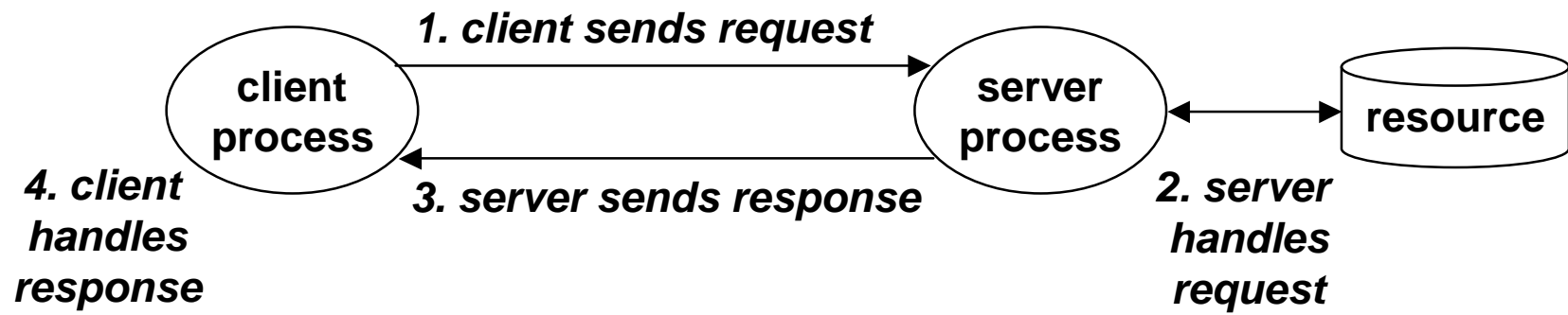
Topics

- Client-server model
- Sockets Interface
- HTTP
- Serving static content
- Serving dynamic content
- Reading 12.7, 12.8 (Beta) or 12.6-12.7 (New)
- Problems: 12.14 (Beta) or 12.10 (New)

Client-server model

Every network application is based on the client-server model:

- Application is a *server* process and one or more *client* processes
- Server manages some *resource*, and provides *service* by manipulating resource for *clients*.
- Client makes a request for a service
 - request may involve a conversation according to some server protocol
- Server provides service by manipulating the resource on behalf of client and then returning a response



Clients

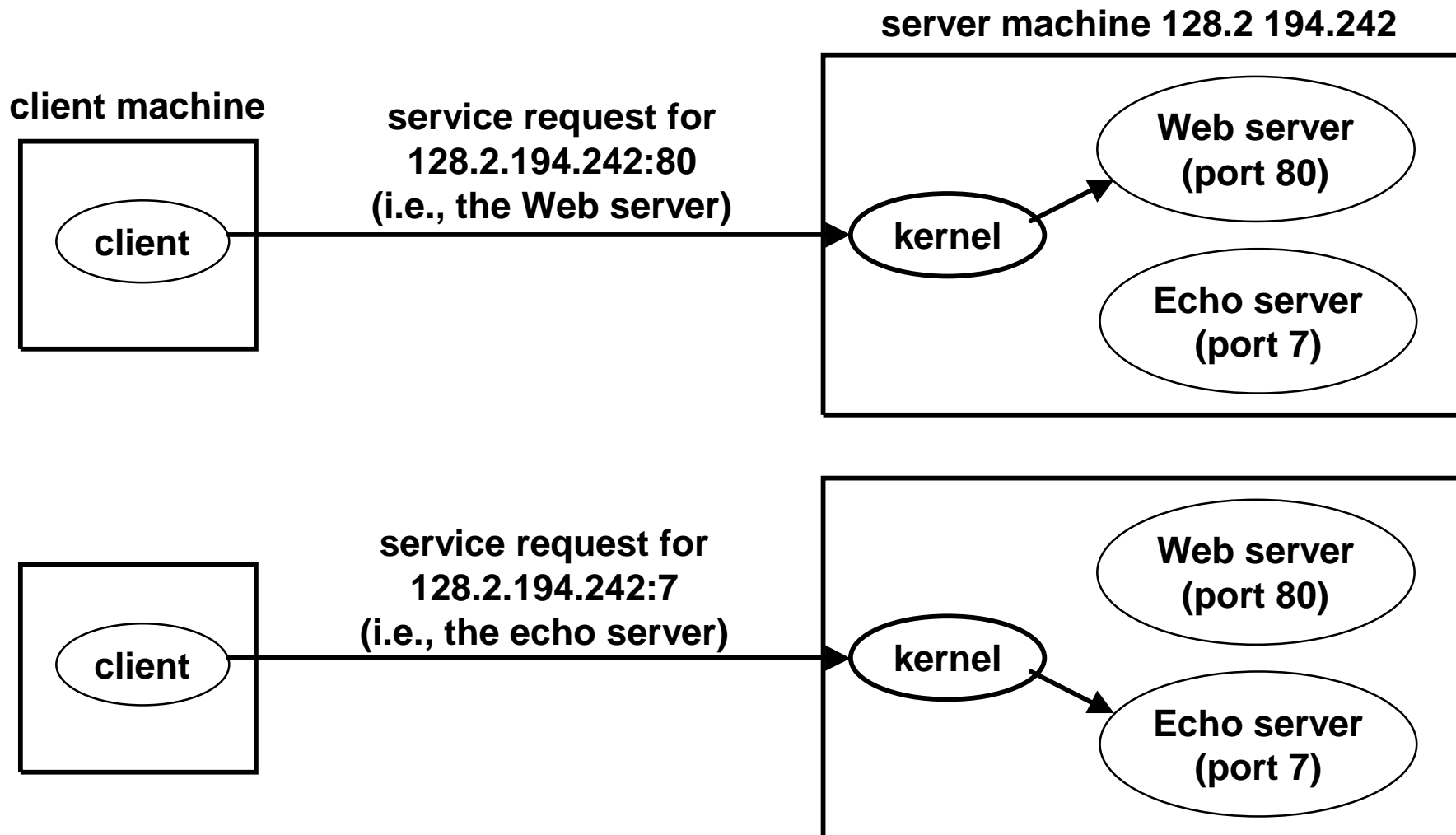
Examples of client programs

- Web browsers, ftp, telnet, ssh

How does the client find the server?

- The address of the server process has two parts: *IPaddress:port*
 - The *IP address* is a unique 32-bit positive integer that identifies the host (adapter).
 - » dotted decimal form: $0x8002C2F2 = 128.2.194.242$
 - The *port* is positive integer associated with a service (and thus a server process) on that machine.
 - » port 7: echo server
 - » port 23: telnet server
 - » port 25: mail server
 - » port 80: web server

Using ports to identify services



Servers

Servers are long-running processes (daemons).

- Created at boot-time (typically) by the init process (process 1)
- Run continuously until the machine is turned off.

Each server waits for requests to arrive on a well-known port associated with a particular service.

- port 7: echo server
- port 25: mail server
- port 80: http server

A machine that runs a server process is also often referred to as a “server”.

Server examples

Web server (port 80)

- resource: files/compute cycles (CGI programs)
- service: retrieves files and runs CGI programs on behalf of the client

FTP server (20, 21)

- resource: files
- service: stores and retrieve files

Telnet server (23)

- resource: terminal
- service: proxies a terminal on the server machine

Mail server (25)

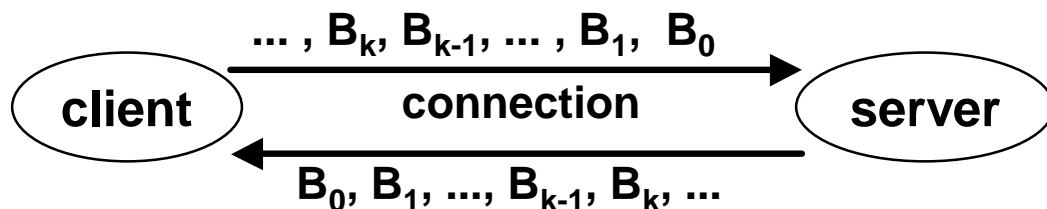
- resource: email “spool” file
- service: stores mail messages in spool file

See `/etc/services` for a comprehensive list of the services available on a Linux machine.

The two basic ways that clients and servers communicate

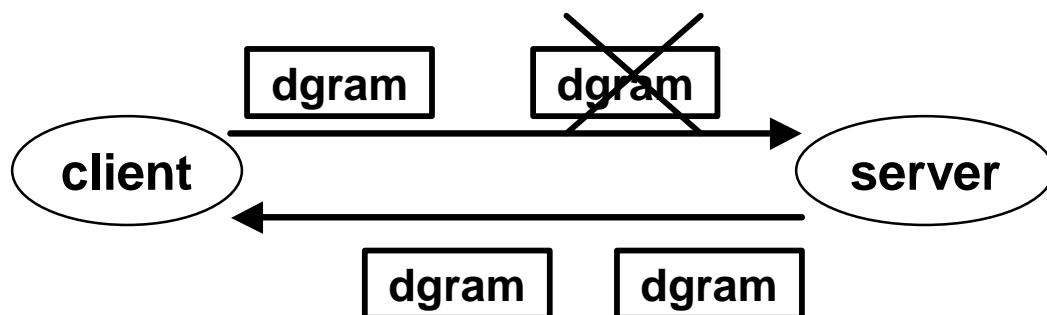
Connections (TCP):

- reliable two-way byte-stream.
- looks like a file.
- akin to placing a phone call.
- slower but more robust.



Datagrams (UDP):

- data transferred in unreliable chunks.
- can be lost or arrive out of order.
- akin to using surface mail.
- faster but less robust.



We will only discuss connections.

Internet connections (review)

Clients and servers communicate by sending streams of bytes over *connections*:

- point-to-point, full-duplex, and reliable.

A *socket* is an endpoint of a connection

- *Socket address* is an `IPAddress:port` pair

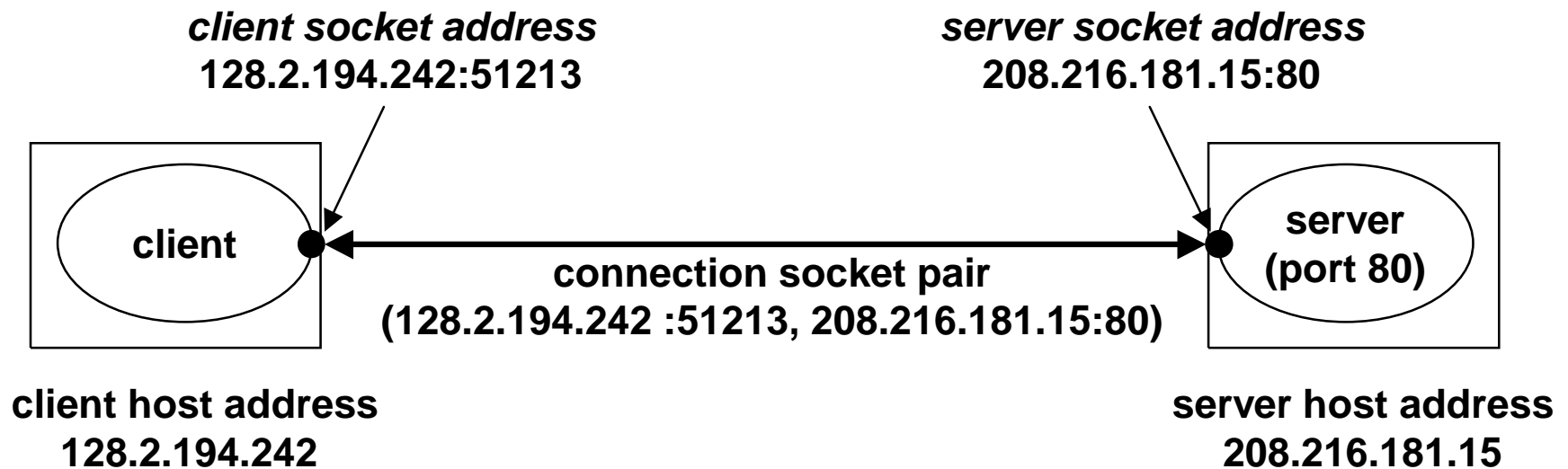
A *port* is a 16-bit integer that identifies a process:

- *ephemeral port*: assigned automatically on client when client makes a connection request
- *well-known port*: associated with some service provided by a server (e.g., port 80 is associated with Web servers)

A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)

- `(cliaddr:cliport, servaddr:servport)`

Anatomy of an Internet connection (review)



Berkeley Sockets Interface

Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols.

Provides a user-level interface to the network.

Underlying basis for all Internet applications.

Based on client/server programming model.

What is a socket?

A *socket* is a descriptor that lets an application read/write from/to the network.

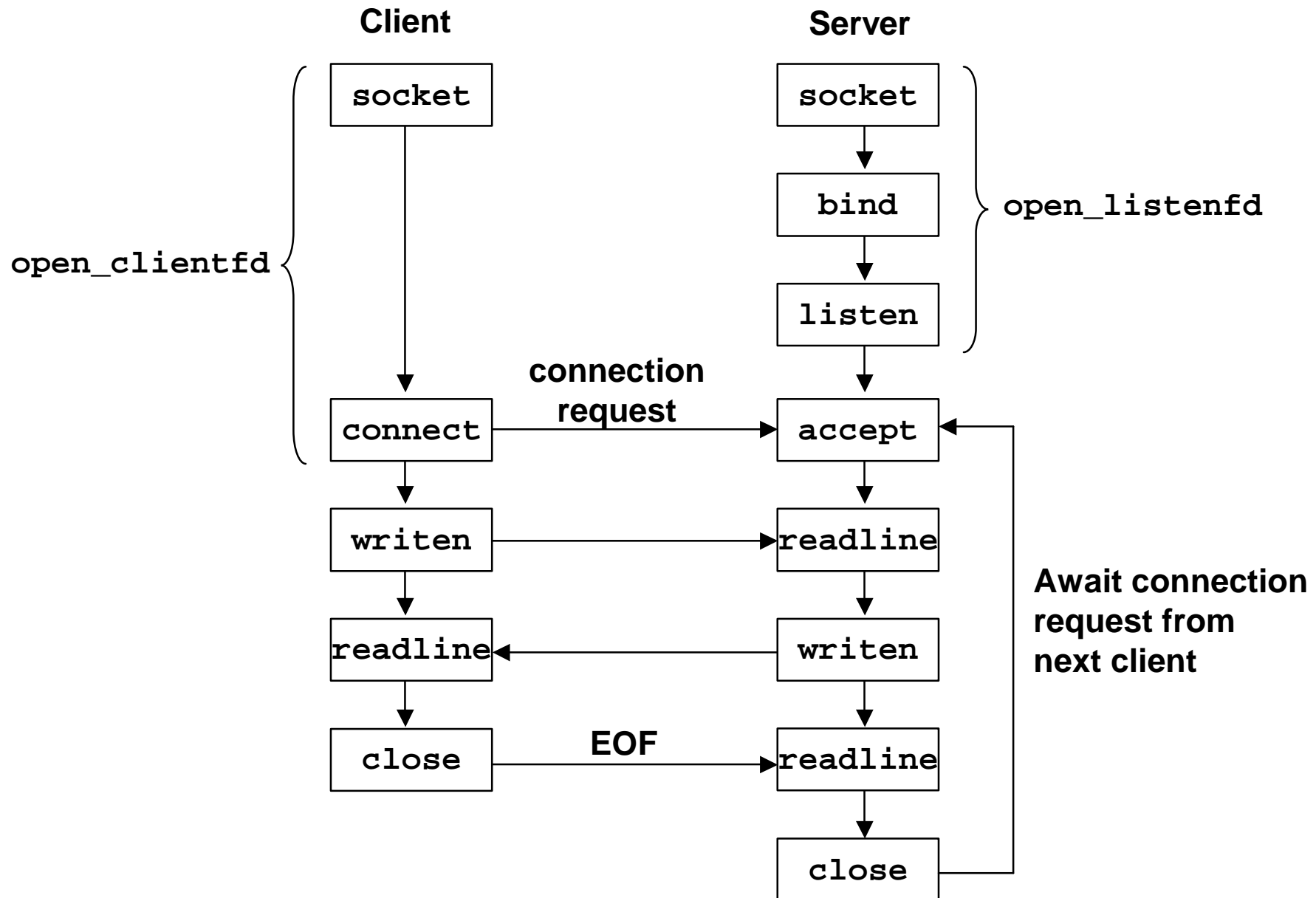
- Key idea: Unix uses the same abstraction for both file I/O and network I/O.

Clients and servers communicate with each by reading from and writing to socket descriptors.

- Using regular Unix `read` and `write` I/O functions.

The main difference between file I/O and socket I/O is how the application “opens” the socket descriptors.

Overview of the Sockets Interface



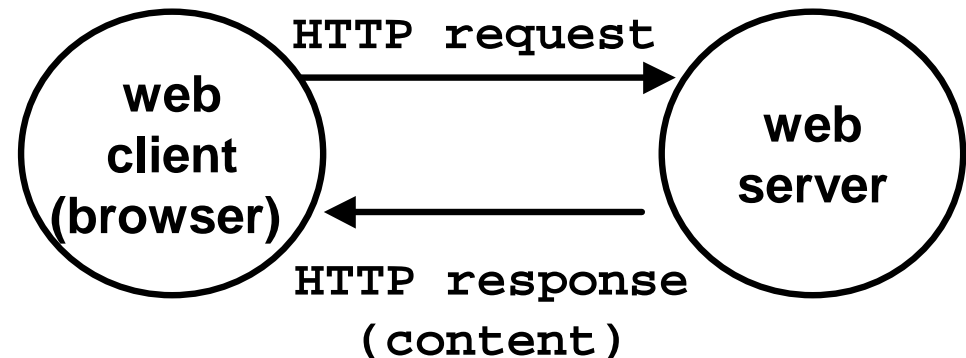
Web servers

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.



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

- | | |
|---------------------------------------|------------------------------------|
| • <code>text/html</code> | HTML page |
| • <code>text/plain</code> | Unformatted text |
| • <code>application/postscript</code> | Postscript document |
| • <code>image/gif</code> | Binary image encoded in GIF format |
| • <code>image/jpg</code> | Binary image encoded in JPG 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.
- **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 content is associated with a file managed by the server.

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.

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 filesystem.
 - 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`).

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

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.

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.

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

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.

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
```

GET response from Apache

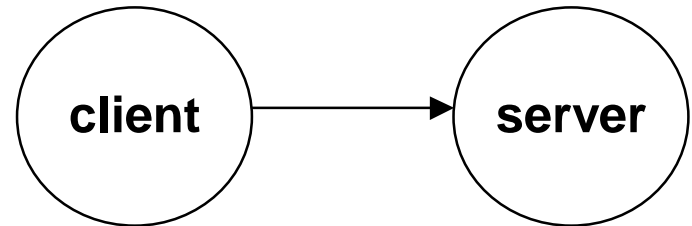
```
HTTP/1.1 200 OK
Date: Thu, 22 Jul 1999 04:02:15 GMT
Server: Apache/1.3.3 Ben-SSL/1.28 (Unix)
Last-Modified: Thu, 22 Jul 1999 03:33:21 GMT
ETag: "48bb2-4f-37969101"
Accept-Ranges: bytes
Content-Length: 79
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
CRLF
<html>
<head>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


Serving dynamic content

Client sends request to server.

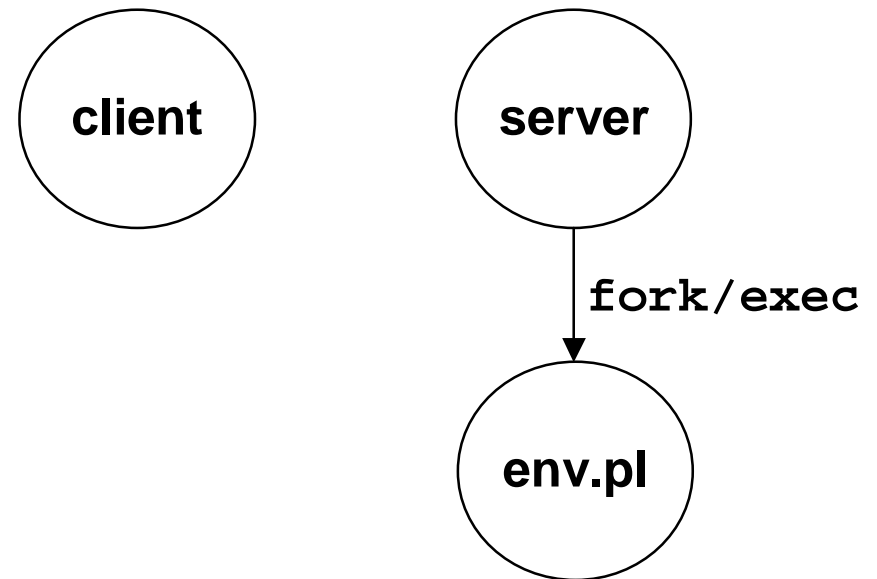
If request URI contains the string “/cgi-bin”, then the server assumes that the request is for dynamic content.

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



Serving dynamic content

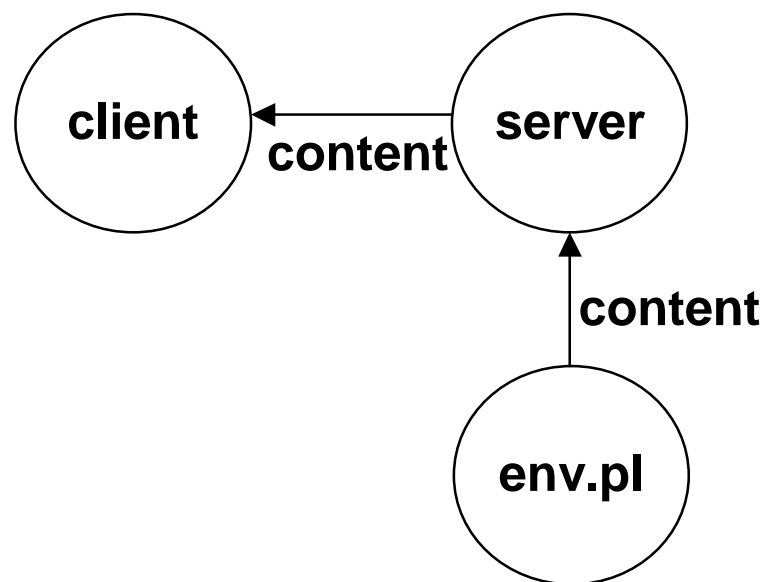
The server creates a child process and runs the program identified by the URI in that process



Serving dynamic content

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

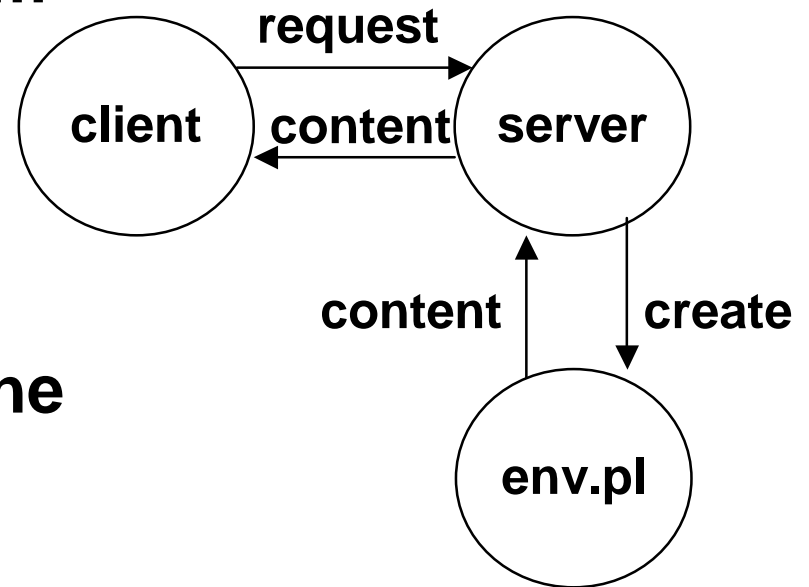
How does the client pass program arguments to the server?

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.

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.

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">
```

Serving dynamic content with GET

URL:

- `http://www.someplace.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.

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);
```

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.

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)

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 “_”)

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);
```

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>
```

HTTP request received by
Tiny Web server

```
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 request sent by client

```
HTTP/1.1 200 OK
Server: Tiny Web Server
```

HTTP response generated by
the server

```
Content-length: 102
Content-type: text/html
<CRLF>
```

HTTP response generated by
the CGI program

```
Welcome to add.com: THE Internet addition portal.
<p>The answer is: 1 + 2 = 3
<p>Thanks for visiting!
```

```
Connection closed by foreign host.
```

```
kittyhawk>
class25.ppt
```

For more information

See 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 JPG images.**
- **220 lines of commented, well structured C code.**
- **Also comes with an implementation of the CGI script for the add.com addition portal.**