Client-server transaction

* Most network applications are based on the client-server model:
  + A server process and one or more client processes
  + Server manages some resource
  + Server provides service by manipulating resource for clinets
  + Server activated by request from client

Sockets

* What is a socket?
  + To the kernel, a socket is an endpoint of communication
  + To an application, a socket is a file descriptor that lets the application read/write from/to the network
    - Remember: all unix I/O devices, including networks, are modeled as files
* Clients and servers communicate with each other by reading from and writing to socket descriptiors

Echo server

* **Lecture 18 slide 6-9**
* The server uses RIO to read and echo text lines until EOF condition is encountered
  + EOF condition is caused by the client calling close (clientfd)
* **Testing the echo server with telnet: slide 10**

Web server basics

* 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)

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
  + Image/gif – binary image encoded in GIF 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
    - Example: 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 file containing executable code
* Bottom line: web content is associated with a file that is managed by the server

URLs and how clients & servers use them

* Unique name for a file: URL (universal resource locator)
* Clients use prefix (ex: http://www.wsu.edu:80) to infer:
  + What kind (protocol) of server to contact (HTTP)
  + Where the server is ([www.wsu.edu](http://www.wsu.edu))
  + What port it is listening on (80)
* Servers use suffix to:
  + Determine if request is for static or dynamic content
    - No hard and fast rules for this
    - One 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)

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)
  + <version> is HTTP version of request
* Request headers: <header name>: <header data>
  + Provide additional information to the server
    - Ex: brand name of the browser or the MIME types that the browser understands

HTTP responses

* HTTP response is a response line followed by zero or more response headers, possibly followed by content, with blank line separating headers from content
* 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: request was handled without error
    - 301: provide alternate URL
    - 404: server couldn’t find the file
* Response headers: <header name>: <header data>
  + Provide additional info about response
  + Content-type: MIME type of content in response body
  + Content-length: length of content in response body

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, PNG, JPEG
  + 239 lines of commented C code
  + Not as complete or robust as a real web server
    - You can break it with poorly-formed HTTP requests

Tiny web server operation

* Accept connection from client
* Read request form client (via connected socket)
* Split into <method> <uri> <version>
  + If method not GET, then return error
* If URI contains “cgi-bin” then sever dynamic content
  + Fork process to execute program
* Otherwise serve static content
  + Copy file to output

Serving static content

* Slide 21 has code

Serving dynamic content

* Client sends request to server
* If request URI contains the string “/cgi-bin”, the tiny server assumes that the request is for dynamic content
* The server creates a child process and runs the program identified by the URI in that process
* The child runs and generates the dynamic content
* The server captures the content of the child and forwards it without modification to the client

Common gateway interface CGI

* Addresses issues with serving dynamic content
* Because the children are written according to the CGI spec, they are often called CGI programs
* CGI defines a simple standard for transferring information between the client (browser), the server, and the child process
  + If codes are written according to the CPI spec, they are often called CGI programs
* CGI is the original standard for generating dynamic content
  + Avoid having to create process on the fly (expensive and slow)

The add.com experience

* Slide 27

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
* 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”
* URL suffix:
  + Cgi-bin/adder?15&18
* Result displayed on browser:
  + Welcome to add.com. The answer is 15 + 18 = 33
* 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 = “15&18”
* 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
  + Code on slide 31
* **Example terminal screenshot on slide 33**

PA 4

* Read CSAPP chapter 11 (especially 11.4-11.6)