Csci 4131
HTTP Revisited
Toward A Simple HTTP Server Built in Python

Lecture 12, February 26th

Spring 2024

Dr. Dan Challou

Logistics – Csci 4131 Lecture 12, February 26th

- HW 3 due Sunday, March 3rd at 11:59pm. Special late submission policies and penalties apply – see the assignment write-up for details. There will be no instructional support over break (we are on break too)

- zyBooks HW 6 is available in your zyBook and due on March 10th

- HW 4 will be issued before we head over for Spring Break

- Exam 1 Results will available on GradeScope tonight or Tomorrow – The regrade request window is open for 7 days after we post grades.

GradeScope

- You have been enrolled
- You should have received an email with information on how to login to gradescope in your university email:

  yourX.500id@umn.edu

  (for example mine is: chal0006@umn.edu)

If you did not get an email with your gradescope information, email the help email (csci4131s24s1-help@umn.edu) or go to an office hour

Note

- Solutions to HW assignments are available and can be reviewed at any office hour, BUT
  - **you will have to take notes.**
- We don't make electronic copies available, and you aren't allowed to take pictures or screen shots without our permission

Reading - HTTP Protocol (HW 4 Refs)

- Foundation of the WWW
- Target – impart a deeper understanding of how the HTTP protocol works, along with a better understanding of browser and web server function
- **Reading:**
  - https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

(The file named: In Introduction to HTTP Basics.pdf in the Resources section of the class Canvas site)

  - RFC 2616 (HTTP 1.1)

https://tools.ietf.org/html/rfc2616

http://www.w3c.org/Protocols/

https://www.jmarshall.com/easy/http/ (This is a nice site too)

Upcoming Reading and Tutorials: Node.js, JSON, Ajax

- Your Zybook – just search for the topics bulleted below
- Node.js
  - https://www.w3schools.com/nodejs/default.asp
  - https://www.tutorialspoint.com/nodejs/
  - https://nodejs.org/en/docs/guides
- JSON
  - https://www.w3schools.com/js/js_json_intro.asp
  - https://www.json.org/
  - Optional: Sebesta – Chapters 10, Section 3.3

- AJAX
  - https://www.w3schools.com/xml/ajax_intro.asp
  - Optional: Sebesta – Chapter 10

Last Time

- URL Wrap-up
- HTTP Request and Response Messages in Detail

Today -

- Lecture 11 Exercises Reviewed
- The HTTP POST Method
- Building and testing a HEAD request (not required)
- Overview, Tips, etc.
- And, if time, Intro to Node.js

Questions?

Lecture 11, Exercise 1:

Specify which of the following items is a URI, URL***, and/or*** URN (URN is defined as Universal Resource Name).

a) isbn:0451450523

b) http://www.cs.umn.edu/academics/classes.php

c) telnet://195.0.1.12:80/

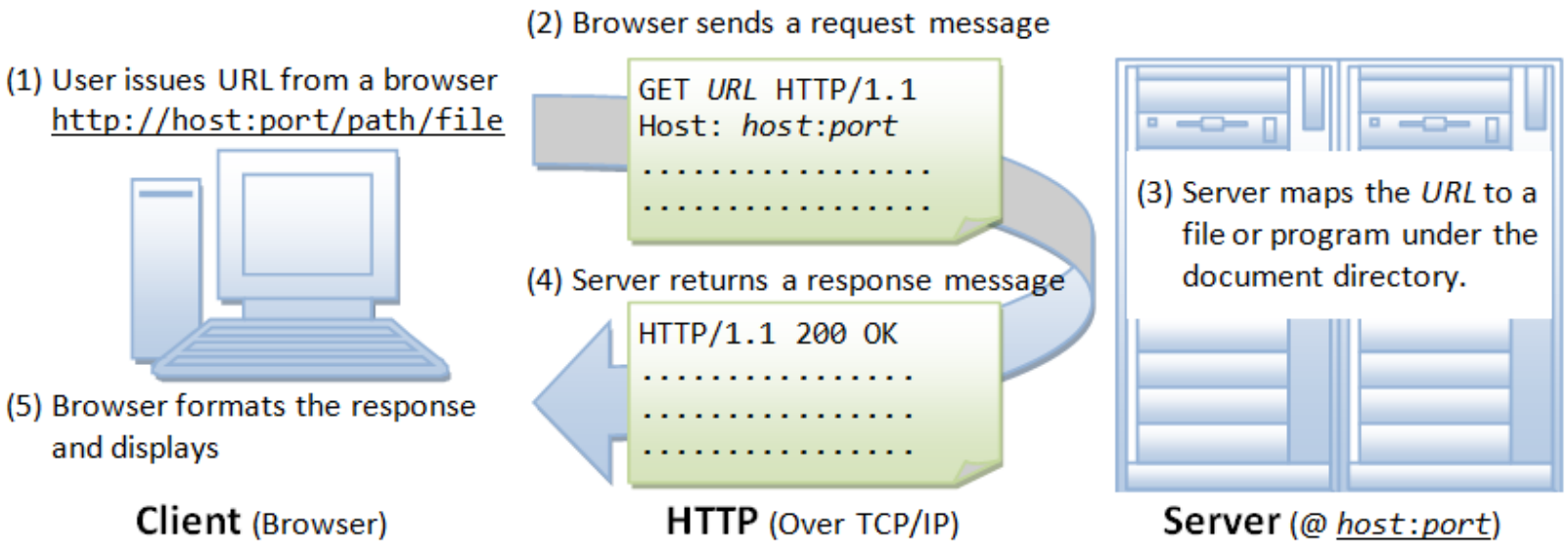
d) mailto:Jim.Smith@acme.com

e) tel: 19146286389

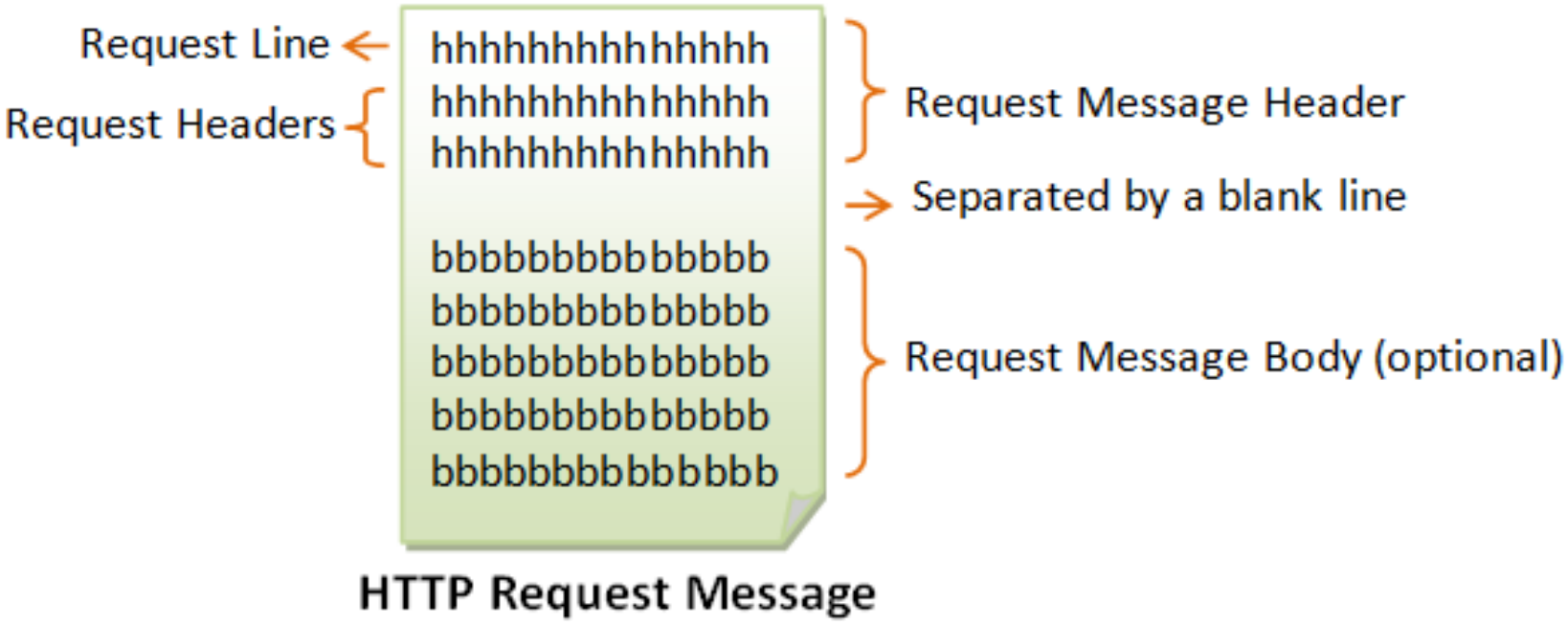
Questions?

Lets Review

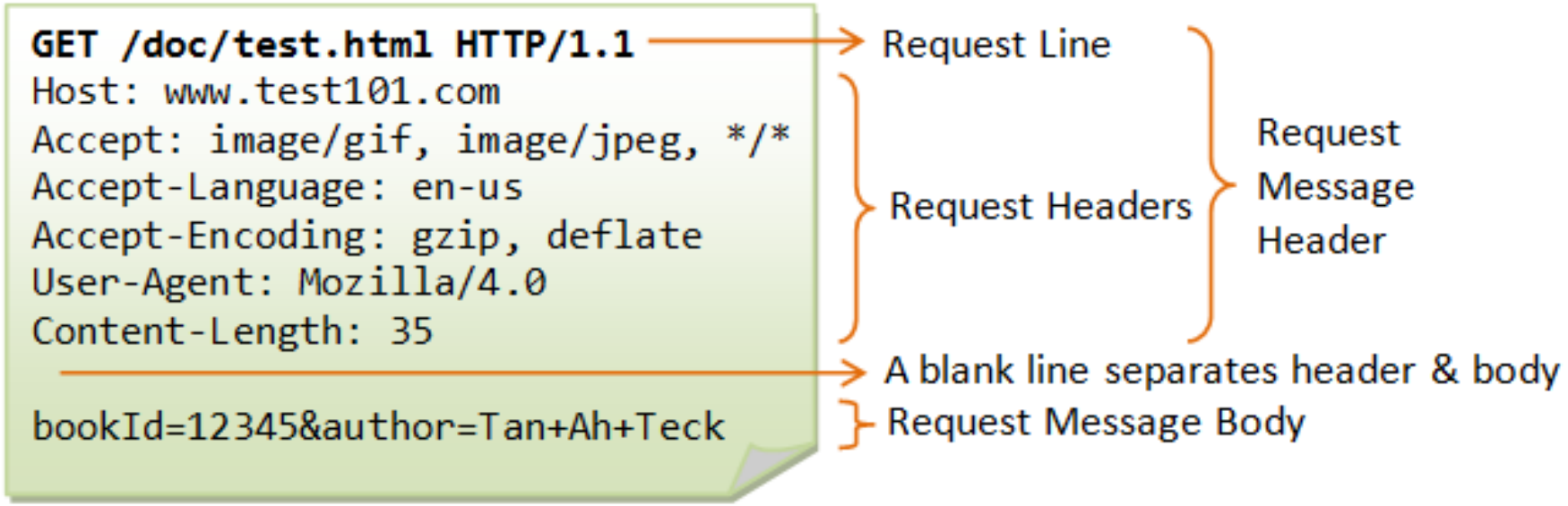
Recall, yet again, how the HTTP Protocol Works



Format of HTTP Request Message



HTTP Request Message Example



http://www.test101.com/doc/test.html?bookId=12345&author=Tan-Ah-Tek

HTTP Request – The Request Line (First Line in the Request)

The first line is the REQUEST LINE, and it contains three items:

1. Name of the requested operation.
2. Request-URL (relative URL - specifying the resource).
3. HTTP version.

In HTTP, message communication is built upon MIME (Multipurpose Internet Message Extension) format.

HTTP Request Methods

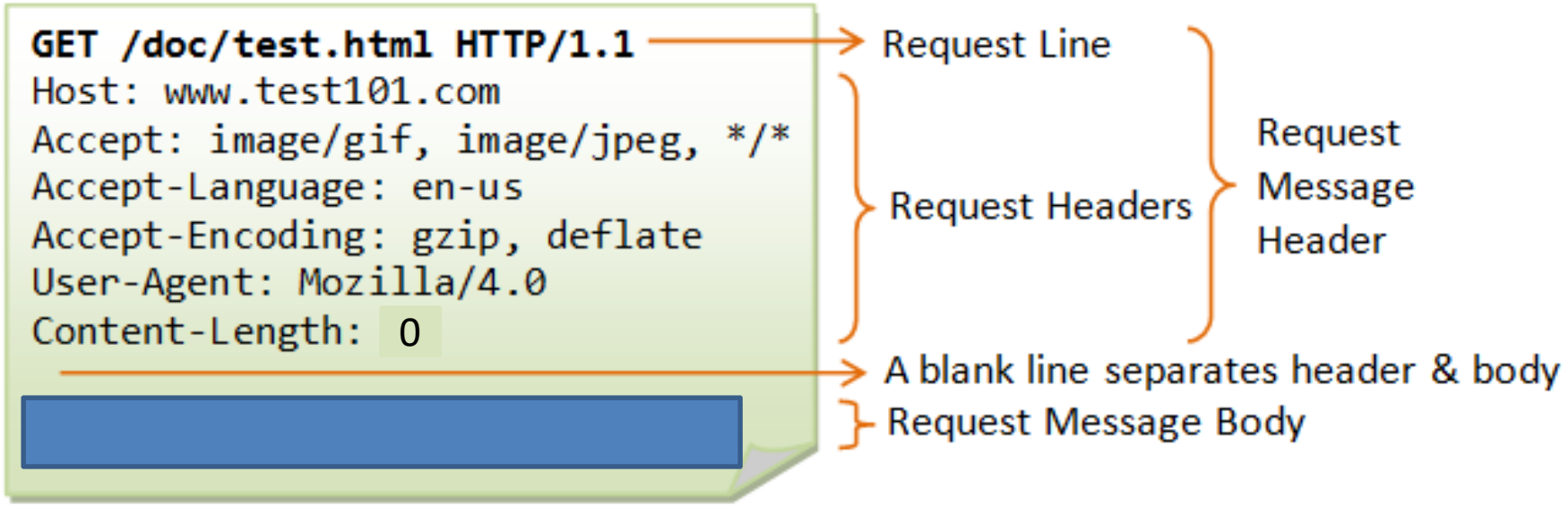
- GET: A client can use the GET request to get a resource from the web server.
- HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- POST: Used to post data up to the web server.
- PUT: Ask the server to store the data.
- DELETE: Ask the server to delete the data.
- TRACE: Ask the server to return a diagnostic trace of the actions it takes.
- OPTIONS: Ask the server to return the list of request methods it supports.
- CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods

Example: User types the following into Browser Address bar: (Assume browser using HTTP/1.1)

- http://www.test101.com/doc/test.html

  (**note**, the URL above does not exist, we are using it for the purposes of instruction)

Browser Generates Something like the following HTTP Request Message



Message Arrives at Server

- Server parses the message
- Checks the Request Method (GET)
- Checks the URI

  i) Does file exist? Yes, go to ii, else error

  ii) Does file have proper access permissions, yes, go to iii, else error

  iii) Compose response message indicating success and get the file, append to headers, encode the message, and send it

  If error, send back message with error response code

Assume that everything is good on the server's end

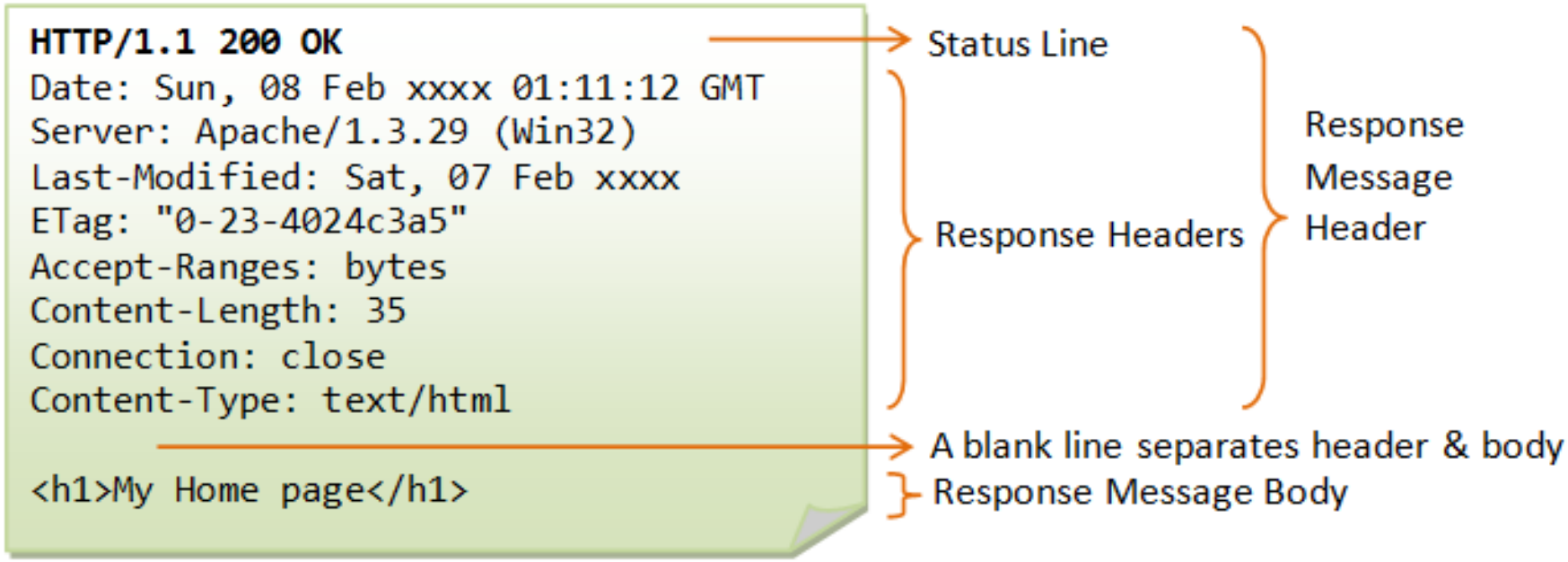
Recall the original request in the browser address bar

http://www.test101.com/doc/test.html

The content of the file: **test.html** is

<h1>My Home page</h1>

The Server Responds with a message something like the following:



Status Line in Response Message (First Line in the Response Message)

- The first line is called the *status line*, followed by optional response header(s).

- The status line has the following syntax:

  HTTP-version status-code reason-phrase

  - *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
  - *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
  - *reason-phrase*: gives a short explanation to the status code.

- Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

- Examples of status line in an HTTP response messge are:

```
HTTP/1.1 200 OK
HTTP/1.0 404 Not Found
HTTP/1.1 403 Forbidden
```

Response Status Codes (on HTTP Response Message Status Line)

- The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.
- The status code is a 3-digit number:

  1xx (Informational): Request received, server is continuing the process.

  2xx (Success): The request was successfully received, understood, accepted and serviced.

  3xx (Redirection): Further action must be taken in order to complete the request.

  4xx (Client Error): The request contains bad syntax or cannot be understood.

  5xx (Server Error): The server failed to fulfill an apparently valid request.

Response Status Codes including Error Codes

- 100 Continue: The server received the request and in the process of giving the response.
- 200 OK: The request is fulfilled.
- 301 Move Permanently: The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called Location. The client should issue a new request to the new location. Application should update all references to this new location.
- 302 Found & Redirect (or Move Temporarily): Same as 301, but the new location is temporarily in nature. The client should issue a new request, but applications need not update the references.
- 304 Not Modified: In response to the If-Modified-Since conditional GET request, the server notifies that the resource requested has not been modified.
- 400 Bad Request: Server could not interpret or understand the request, probably syntax error in the request message.
- 401 Authentication Required: The requested resource is protected, and require client's credential (username/password). The client should re-submit the request with his credential (username/password).
- 403 Forbidden: Server refuses to supply the resource, regardless of identity of client.
- 404 Not Found: The requested resource cannot be found in the server.
- 405 Method Not Allowed: The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested.
- 406 Not Acceptable
- 408 Request Timeout:
- 414 Request URI too Large:
- 500 Internal Server Error: Server is confused, often caused by an error in the server-side program responding to the request.
- 501 Method Not Implemented: The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get").
- 502 Bad Gateway: Proxy or Gateway indicates that it receives a bad response from the upstream server.
- 503 Service Unavailable: Server cannot response due to overloading or maintenance. The client can try again later.
- 504 Gateway Timeout: Proxy or Gateway indicates that it receives a timeout from an upstream server.

Example of a Request Message that would result in a 406 Not Acceptable Error Code

GET /docs/mypic.png HTTP/1.1

Host: www.test101.com

Accept: image/gif, image/jpeg

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

(blank line)

- **Why????**

Notes – for most Web Servers in real use

- The request method name "GET" is case sensitive and must be in uppercase.
- If the request method name was incorrectly spelled, the server would return an error message "501 Method Not Implemented".
- If the request method name is not allowed, the server will return an error message "405 Method Not Allowed". For example, DELETE is a valid method name, but may not be allowed (or implemented) by the server.
- If the *request-URL* does not exist, the server will return an error message "404 Not Found". You have to issue a proper *request-URL*, beginning from the document root "/". Otherwise, the server would return an error message "400 Bad Request".
- If the *HTTP-version* is missing or incorrect, the server will return an error message "400 Bad Request".
- In HTTP/1.0, by default, the server closes the TCP connection after the response is delivered. If you use telnet to connect to the server, the message "Connection to host lost" appears immediately after the response body is received. You could use an optional request header "Connection: Keep-Alive" to request for a persistent (or keep-alive) connection, so that another request can be sent through the same TCP connection to achieve better network efficiency. On the other hand, HTTP/1.1 uses keep-alive connection as default.

Exercise 2: Submit your answers via the Lecture 12, Exercise 2 Submission item on Canvas (raise hand when done)

An HTTP 1.1. Compliant Python webserver is running on the host computer:

`csel-kh1262-11.cselabs.umn.edu.`

The server was executed (i.e., run) from the `/webserver` folder (directory) (which has world executable permissions) and is listening on port `9004`. The contents of `the /webserver` folder (directory) are as follows:

```
-rw------- 1 x500user CSEL-student 3275 Oct 3 07:25 server.py
-rw-r--r-- 1 x500user CSEL-student 1261 Oct 4 17:42 schedule.html
-rw-r--r-- 1 x500user CSEL-student  368 Oct 4 21:40 main.js
-rw-r--r-- 1 x500user CSEL-student 2561 Oct 4 17:42 my schedule.html
```

What HTTP 1.1 *message response status code* will be sent by the server listening on port 9004 to the client/browser after the 3 requests below are sent by the client/browser?

```
GET /webserver/schedule.html HTTP/1.1
Host: csel-kh1262-11.cselabs.umn.edu
Accept: */*

DELETE /webserver/main.js HTTP/1.1
Host: csel-kh1262-11.cselabs.umn.edu
Accept: */*

HEAD /webserver/my schedule.html HTTP/1.1
Host: csel-kh1262-11.cselabs.umn.edu
Accept: */*
```

Unix / Linux: File Permission/Access

- https://www.tutorialspoint.com/unix/unix-file-permission.htm
- https://www.geeksforgeeks.org/permissions-in-linux/

Question (Think/Pair/Share):

- **Can you write the names of one or more programs that receive and respond to HTTP request messages** (30 seconds).

- Then SHARE!!!!

The HTTP POST Request Method

A POST request is used to send data to the server to be processed in some way, like by a node.js server, Python Script, PHP script, or CGI script.

A POST request is different from a GET request in the following ways:

1) There's a block of data sent with the request, in the message body.

2) There are usually extra headers to describe this message body, like **Content-Type** and **Content-Length**

3) The *request URL* is not a resource to retrieve;
it's usually a program/script to handle the data you're sending (and usually the default URL for the site)

4) The HTTP response message normally contains program output, not a static file.
(For HW4, your Python Server will build an HTML page with a table that consists of the name: value pairs sent in the message body)

Consider the following form

- FormPostReqEx.html

- And, suppose the user enters
  - **Tianming Cui** into the field username, and
  - **4131TCta$** into the field password

- Then clicks on Send

Something like the following Post Request Message is generated by the Browser and sent to the Server

POST / HTTP/1.1
Host: 127.0.0.1:8000
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/FormPostReq.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 37
Connection: Keep-Alive
Cache-Control: no-cache

user=Tianming+Cui&pw=4131TCta$

Response – can have a body or not

HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 12:10:12 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Content-Type: text/html

<html><body><h1>It works!</h1></body></html>

So Lets Start to Look At How to Build a Simple HTTP Server

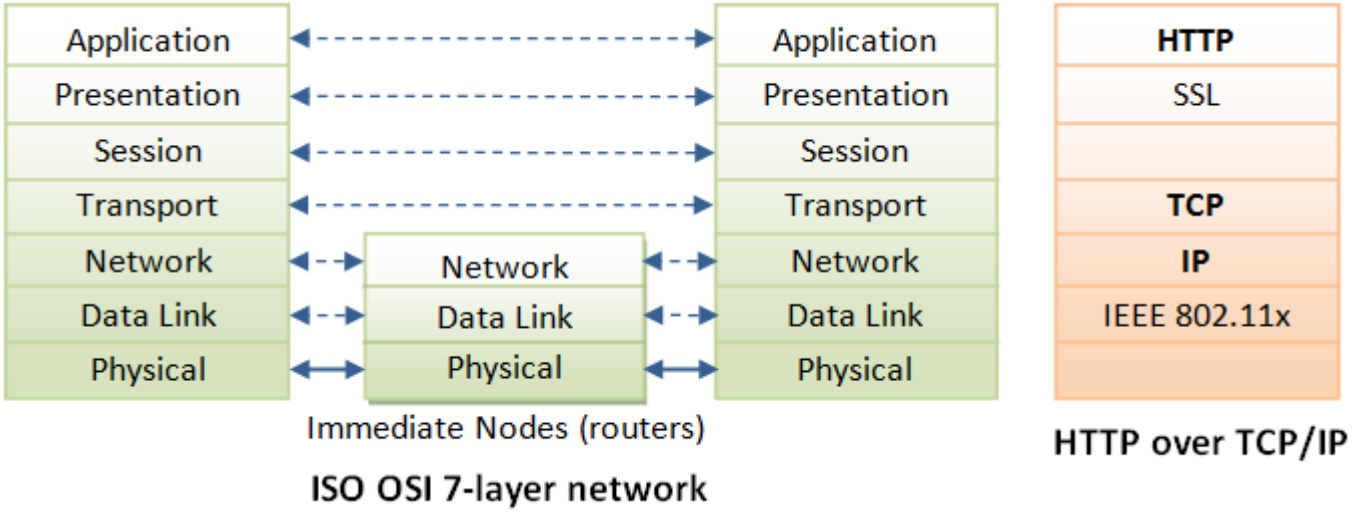
- One that does not use the Python HTTP libraries!!!!

Lets have a look at Python Programs: Echo Client and Echo Server

- ISO-OSI Level 6 Programs and below – can be accessed via sockets (ISO level 6 functionality)
- ***HTTP is an ISO-OSI Level 7 protocol –that is an Application-level protocol***
- HTTP is built on level 6 (and below) applications
- You are free to re-use this echo client and echo server to help construct the Web Server you will build for HW 4

Recall: HTTP is built on Presentation Layer Protocols, which is built on TCP/IP

- HTTP is a client-server application-level protocol.
- It typically runs over a presentation layer protocol with a TCP/IP connection underneath, as illustrated below.



- (***HTTP need not run on TCP/IP. It only presumes a reliable transport. Any transport protocols that provide such guarantees can be used.***)
- See: https://en.wikipedia.org/wiki/Transport_layer for a nice discussion of other possible protocols

Interface to Layer 6 (sockets) Echo Client & Server Code Review

EchoClient.py and Echo Server.py are Available on Canvas in the Lecture 7 Materials,

Please download them now and we will review them

Started with EchoServer.py – will pick up there next time!

Next Class

- More Details on formulating an HTTP Server in Python
- Node.js?

Next Class

- Exam 1 on Wednesday, March 2nd !!