HTTP: HyperText Transfer Protocol

HTTP 0.9

- Used in early 90s ('91-).
- Supported only one command: GET
  - e.g., GET /foo.html
- Supported HTML files only.
  - Did not support images, PS, PDF, music, etc.
- No error handling
- No commands available to send data from a client to a server.
  - An exception: sending a query words to a server
    - GET /cgi-bin/search?umass+boston

The First Web Server

- Built by Tim Berners-Lee
  - CERN httpd (1990)
    - HTTP 0.9 and a simple HTML specification
    - Its source code and other tools (e.g., a browser called "WorldWideWeb") were out in 1991.
- Berners-Lee founded the World Wide Web Consortium (W3C; www.w3.org) in 1994.
  - Standard consortium to standardize Web-related technologies
    - e.g., HTTP, HTML, CSS, XML, DOM, SVG, SOAP, etc.

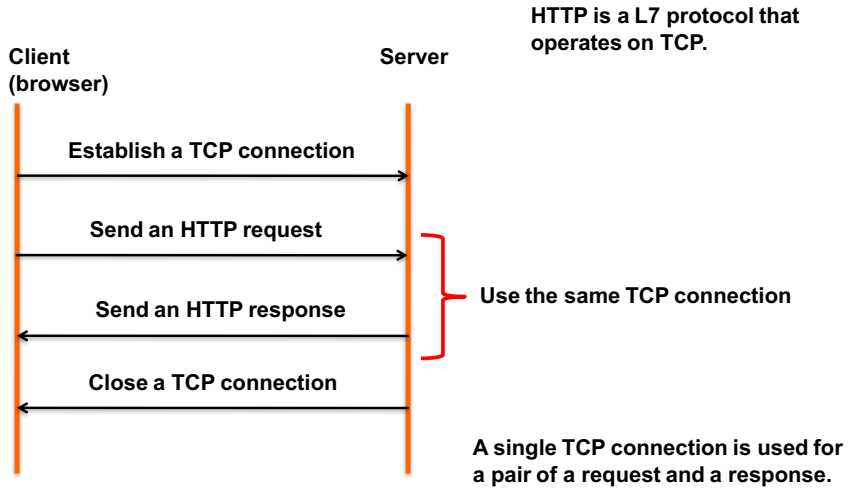
History of Web Browsers

- 1991:
  - Berners-Lee's WorldWideWeb
- 1993:
  - NCSA Mosaic by Marc Andreessen et al.
    - National Center for Supercomputing Applications at UIUC
- 1994:
  - Andreessen co-founded Netscape Communications and released Netscape Navigator
    - Netscape invented critical Web-related tools like SSL, JavaScript, Cookie…
- 1995:
  - Microsoft released Internet Explorer.
  - "Browser war" between Netscape and Microsoft
- 1998:
  - AOL bought Netscape.
  - Mozilla project/organization was started to develop a next-gen Internet tool suite for Netscape.
- 2003:
  - Mozilla Foundation started as a non-profit organization.
    - Firefox, Thunderbird, Bugzilla, etc.

HTTP 1.0

- Developed from 1992 to 1996.
  - The first draft released in 1993.
  - Adopted as an IETF RFC 1945 in May 1996.
    - http://www.ietf.org/rfc/rfc1945.txt
- Several major extensions on HTTP 0.9
  - Structured request-response exchanges
    - HTTP header
    - Extra commands in addition to GET
  - Internet media type
    - Originally called MIME (Multipurpose Internet Mail Extensions) type
    - Sometimes called "content-type"
  - Status code
    - For error handling

HTTP and TCP



An Example HTTP Request

- If you type www.umb.edu in the URL box in your browser,
- Your browser transmits the following HTTP request to www.umb.edu
  - GET / HTTP/1.1
    Host: www.umb.edu:80
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0) Gecko/20100101 Firefox/4.0
    Accept: text/html…
    Accept-Language: en-us,en;q=0.5
    Connection: keep-alive

HTTP Request

- Request = Request-Line
  *(Header)
  <empty line>
  <request body>

  Request-Line = Command SP Request-URI SP HTTP-Version
  - e.g., GET /index.html HTTP/1.0
  - HTTP-Version shows the ability of a client's HTTP handling (the highest version number that a client can handle).
    - A client can state 1.1 in a request and receive HTTP 1.0 responses.

HTTP Response

- Response = Status-Line
  *(Header)
  <empty line>
  <response body>

  Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase

  – HTTP/1.0 200 OK
    Content-Type: text/html
    Date: Tue, 05 Apr 2000 12:00:00 GMT

    <html><body>abc</body></html>

  – HTTP-Version shows the ability of a server's HTTP handling (the highest version number that a server can handle).
    - A server can state 1.1 in a response when it understands HTTP 1.0 and 1.1.

HTTP Commands

- Request = Request-Line
  *(Header)
  <empty line>
  <request body>
  Request-Line = ***Command*** SP Request-URI SP HTTP-Version

- GET
  – Requests Request-URI
    - URI (Uniform Resource Identifier)
  – Receives the content of Request-URI
  – GET /foo.html HTTP/1.0
  – GET /cgi-bin/search?foo HTTP/1.0
  – GET /servlet/helloworld HTTP/1.0

- HEAD
  – Requests the header items of Request-URI
  – Receives the header items only (no content)

  – Head /foo.html HTTP/1.0

  – Web crawlers (bots) often use this command.
    - e.g., checks whether Request-URI exists
      – Can save bandwidth consumption

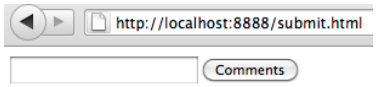
- POST
  – Transmits (arbitrary) data to Request-URI.
    - Need to include Content-Length in a request header.
      – POST /foo/submit HTTP/1.0
        Content-Length: 100

        [100-byte data follows after an empty line]

    - Transmitted data in an request body should belong to (should be understandable by) Request-URI.

HTTP POST Command

- A typical use case:
  - Sending data entered in a form on a HTML file
    - <html>
      <form method="post" action="http://localhost:8888/upload">
      <input type="text' name='status">
      <input type="submit" value="Comments">
      </form>
      </html>



- Request
  - POST /upload HTTP/1.1
    Host: localhost:8888
    User-Agent: Mozilla/5.0 …
    Accept: text/html,…
    Referer: http://localhost:8888/submit.html
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 10 ← Mandatory header field

    status=ZZZ ← Transmitted data (10 bytes)

HTTP Request Header

- Contains zero or more key-value pairs
  - Request = Request-Line
    (Header)
    <empty line>
    <request body>
    Request-Line = Command SP Request-URI SP HTTP-Version

- Major header fields
  - Host
    - IP (or host name) and port number associated with Request-URI that is being requested
    - Host: 127.0.0.1 8888
    - Host: localhost 8888
  - User-Agent
    - Shows a client's (browser's) information
      - User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0) Gecko/20100101 Firefox/4.0
    - A server can collect stats on client browsers/OSes and/or do access control based on browser types.
  - Date
    - Indicates when a request is created.
      - Date: Thu, 07 Apr 2011 20:13:59 GMT

HTTP Response: Status Code

- Response = Status-Line
  *(Header)
  <empty line>
  <response body>

  Status-Line = HTTP-Version SP ***Status-Code SP Reason-Phrase***

- Status code categories
  - 200 level:
    - A server has received and processed a request successfully.
  - 400 level:
    - An error occurred because a request has a syntax error(s) or a server denies to process a request.
  - 500 level:
    - An error occurred when a server processes a request.

- Representative status code
  - 200: OK
    - Everything went well.
    - HTTP/1.0 200 OK
  - 400: Bad Request
    - A request has a syntax error in Request-URI.
      - e.g., GET index.html HTTP/1.0
        - » URL should have been /index.html
  - 401: Unauthorized (authorization required)
    - A request is made for password-protected Request-URI.
    - Wrong user name and/or password were supplied.
  - 404: Not Found
    - Request-URI does not exist.
  - 500: Internal Server Error
    - A server encounters an unexpected error and cannot process a request
  - 501: Not Implemented
    - A server does not implement the command in a request.
  - 503: Service Unavailable
    - A server cannot process a request due to a temporary overloading or maintenance.

HTTP Response Header

- Response = Status-Line
  (Header)
  <empty line>
  <response body>

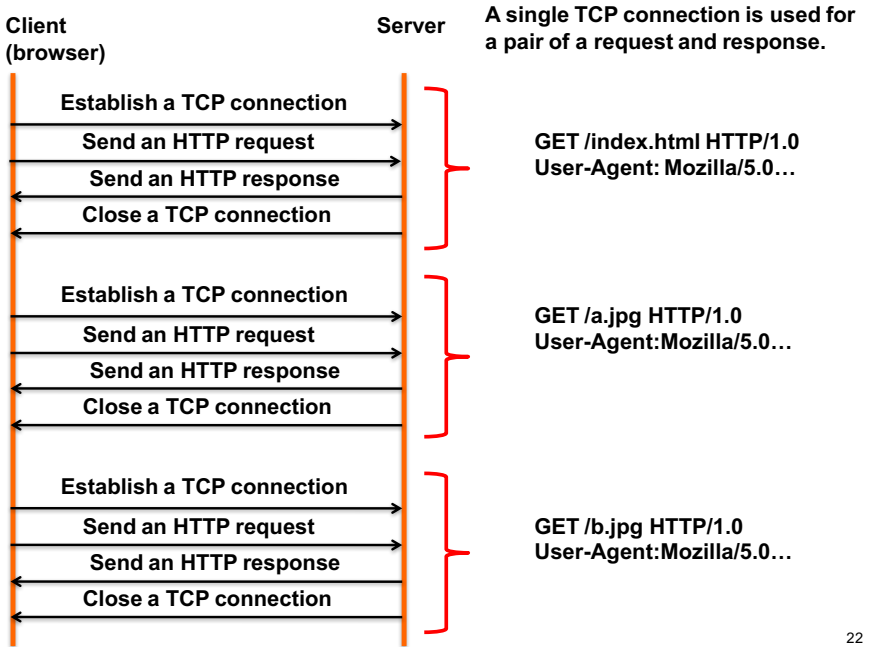
  Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase

- Major header fields
  - Date
    - Indicates when a response is created.
      - Does not indicate when Request-URI is created/updated.
    - Date: Thu, 07 Apr 2011 20:13:59 GMT
  - Server
    - Shows a server's information
    - Server: Apache/2.2.16 (Ubuntu) PHP/5.3.3-1ubuntu9.1

  - Content-Length
    - Indicates the size of a response body.
    - Content-Length: 121
      - 121 bytes
  - Content-Type
    - Indicates the media type of a response body.
    - Content-Type: text/html
  - Last-Modified
    - Indicates the timestamp of a response body (Request-URI)

Connection Management

- Connection-per-resource
  - With HTTP 1.0, a client (browser) sets up HTTP connections on a resource-by-resource basis.
    - If a client requests an HTML file that contains 2 images (2 <img> tags), it sets up 3 separate (TCP) connections.
      - <html>
        <body>
        Testing TinyHttpd…
        <p><img src="a.jpg">
        <p><img src="b.jpg">
        </body>
        </html>

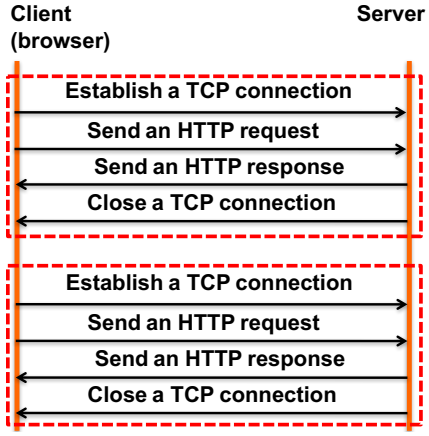


- By the way… in this case,
  - <html><body> Testing TinyHttpd…
    <p><img src="a.jpg">
    <p><img src="a.jpg">
    </body></html>

- Some browsers set up 2 connections (not 3) to reduce bandwidth consumption.
  - However, this is not a mandatory behavior of HTTP 1.0.
  - Your browser may set up 3 connections.

HTTP: A Stateless Protocol

- Stateless (not stateful)
  - Keeps no "states" across different request-response pairs.



- Request-response pairs do not know with each other.
  - A pair does not know what a previous pair did.
  - A pair has no impacts on the behavior of future pairs.
- No shared data across different pairs.
  - Exception: HTTP cookie
    - a method to share data across different request-response pairs.

Timeout on accept() and read()

- java.net.SocketOptions.SO_TIMEOUT
  - Timeout in milliseconds for blocking operations on a socket (Default: 4102)
    - SocketInputStream.read() and ServerSocket.accept()
- Socket.setSoTimeout(int)
  - Sets the timeout period for SocketInputStream.read()

```
ServerSocket serverSocket = new ServerSocket(…);
Socket client = serverSocket.accept();
client.setSoTimeout(30000);          // 30 seconds
client.getInputStream().read();      // Read a request
```

- ServerSocket.setSoTimeout(int)
  - Sets the timeout period for ServerSocket.accept()

```
ServerSocket serverSocket = new ServerSocket(…);
serverSocket.setSoTimeout(60000);    // 60 seconds
Socket client = serverSocket.accept();
```

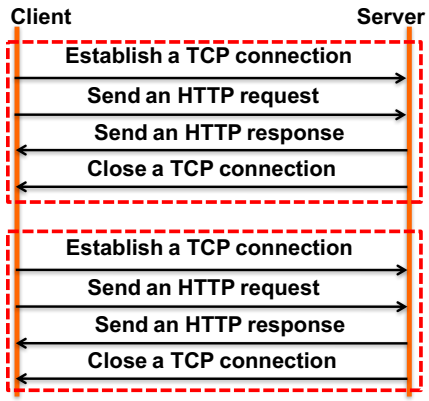
- SocketTimeoutException is thrown, if a timeout occurs.

HW 28-1

- Run java edu/umb/cs/threads/tinyhttpd/TinyHttpd2 on a terminal.
- Type “http://localhost:8888/” in a web browser on the same machine.
  - The web server’s terminal shows all information (i.e., command and header) that the web browser sends.
  - The web browser displays index.html.
- Run telnet on another terminal to access the web server, and then type an HTTP command.
  - telnet localhost 8888
  - GET /index.html HTTP/1.0, followed by empty line (carriage return)
  - The web server’s terminal shows the HTTP command.
  - The telnet terminal shows the content of index.html.

- Modify TinyHttpd2.executeCommand()
  - Parse an HTTP request message
    - with java.util.StringTokenizer or java.util.regex.*
  - Retrieve arbitrary HTML files and image files
    - e.g., JPEG, PNG, etc.
    - Support at least one type of image files
- Confirm the connection-per-resource policy in HTTP 1.0 with TinyHttpd2.
  - <html><body> Testing TinyHttpd…
    <p><img src=”a.jpg">
    <p><img src=”b.jpg">
    </body></html>
- Confirm the client/browser displays text and images correctly.

- Implement TinyHttpd3 by extending TinyHttpd2
  - Process a request-response pair with an extra thread (thread-per-connection policy)

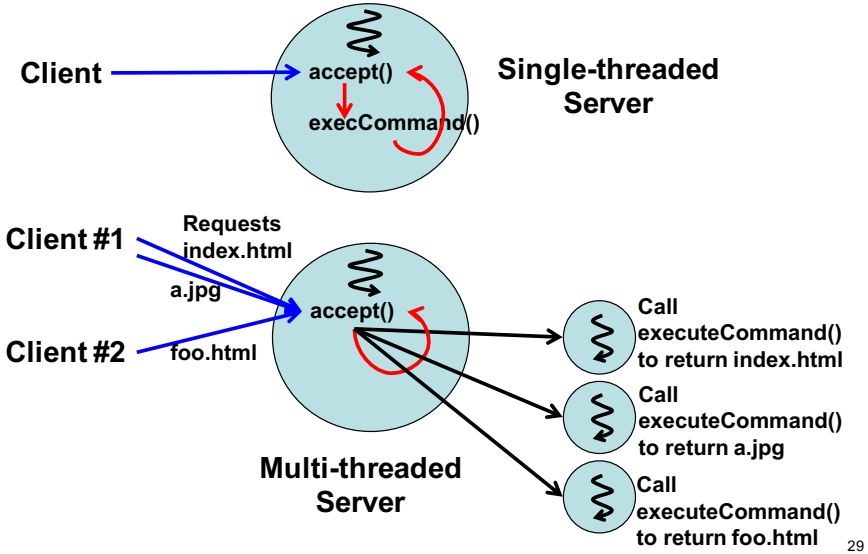


Modify TinyHttpd2.init()

```
while(true){
  Socket client =
    serverSocket.accept();
  executeCommand( client );
}

while(true){
  Socket socket =
    serverSocket.accept();
  new Thread(
    new YourRunnable(…)).start();
}
```

Note: Thread-per-connection + connection-per-resource = thread-per-resource



- Implement a subset of HTTP 1.0.
  - Commands: GET, HEAD and POST
    - Return 200 (OK) or 404 (Not Found) status code for each command.
    - Return the 501 (Not Implemented) error status when receiving other commands.
  - Response header fields
    - Date, Server, Content-Length, Content-Type (text/html), Last-Modified
- Test TinyHttpd3 with telnet and a web browser.

- Support HTTP POST
  - Test it with your browser

```
<html>
 <form method="post" action="http://localhost:8888/upload">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
 </form>
</html>
```

First name:
Last name:
Submit

- No need to implement the "action" part in the server.
  - Your web server can simply ignore it.
  - At the server side, it should print out data transmitted by a client.

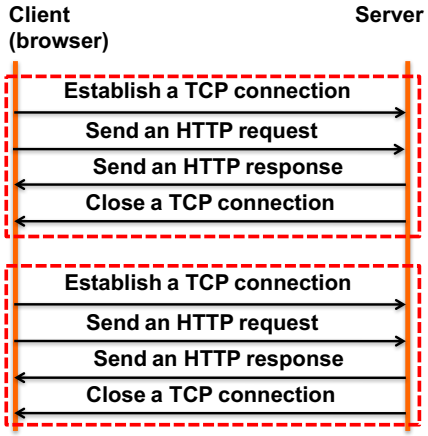
HTTP 1.1

- A revision of HTTP 1.0 started in W3C around 1995.
- Completed in 1999 (RFC 2616)
  - http://www.w3.org/Protocols/rfc2616/rfc2616.html
  - http://www.w3.org/Protocols/
- Major extensions on HTTP 1.0
  - Extra HTTP commands and header fields
  - Persistent connections
  - Protocol-level caching
  - Virtual hosts
  - Digest authentication
    - in addition to the basic authentication, which is a part of HTTP 1.0

Persistent Connections

- HTTP 1.0
  - Connection-per-resource
    - A single TCP connection is used to transmit a pair of a request and a response.
    - Multiple connections are used to transmit multiple request-response pairs.
- HTTP 1.1
  - Connection-per-client
    - Allows a single connection to transmit multiple request-response pairs.
    - Aims for performance improvement.

Recap: HTTP and TCP



HTTP is a L7 protocol that operates on TCP.

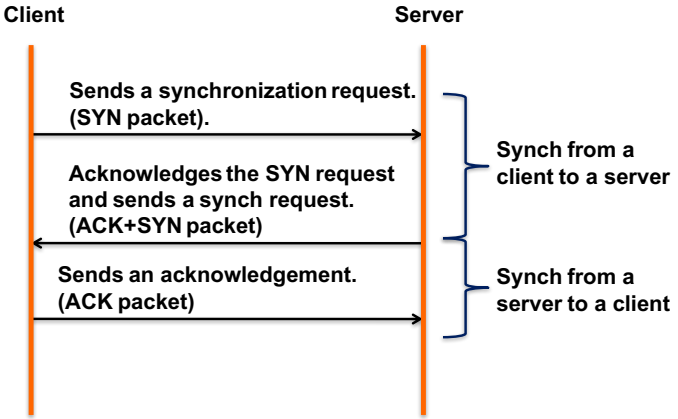
A single TCP connection is used for a pair of a request and response.

What's Wrong in Connection Mgt in HTTP 1.0?

- TCP connection establishment and termination are expensive.
- More HTTP requests → more TCP connections
  - Higher per-resource latency
    - Longer download and page-loading time
  - Higher workload in servers.
    - Higher CPU workload.
    - A server may run out of file descriptors when it handles many simultaneous clients.
  - Higher bandwidth consumption

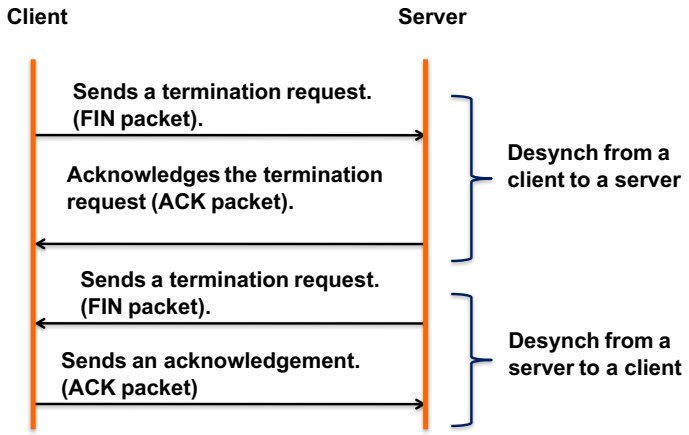
TCP Connection Establishment

- Three-way handshake



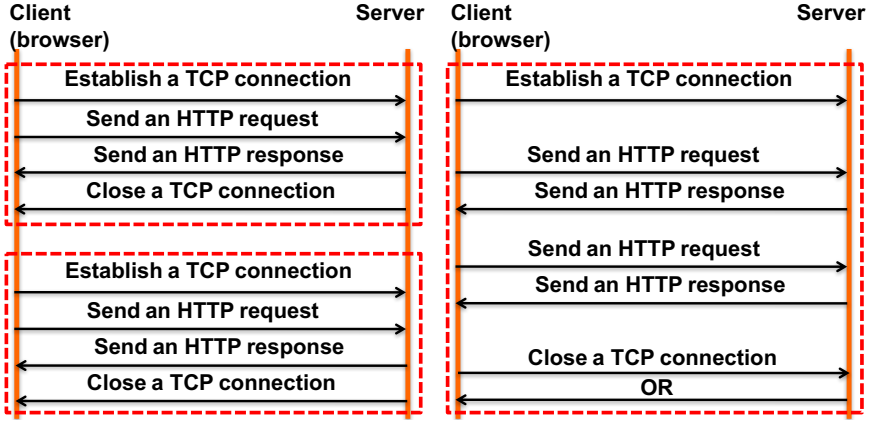
TCP Connection Termination

- Either a client or a server can start closing a connection.



Connection-per-client (Persistent Connection) Policy

- Once a TCP connection is established, it is reused by multiple HTTP request-response pairs.
  - Reduces client-side latency, server-side workload and bandwidth consumption



Connection Header Field

- HTTP 1.1 assumes all connections are *persistent* by default.
- In practice…
  - Most 1.1-compliant browsers explicitly send the "Connection" header field to servers.
    - Connection: keep-alive
    - because there still exist many HTTP 1.0 servers.
  - Most 1.1-compliant servers explicitly send the "Connection" header field to clients if they receive it in requests.
    - If a request states HTTP version 1.0 and does not have the "Connection" header, a server does not send the "Connection" header and should go for the connection-per-resource policy.
  - HTTP 0.9/1.0 servers can handle persistent connections (send "Connection" header field) when they receive it from clients.

Max # of Connections Per Server

- The maximum number of connections a client can maintain for a single server
  - The HTTP spec says 2.
  - Widely-accepted practice
    - IE6, IE7: 2
    - IE8, IE9, Firefox, Chrome, Safari: 6

Connection Termination

- A connection will keep open until either the client-side or server-side closes it.
  - A client may want to close a connection
    - when the # of connections reaches the maximum.
      - Connection replacement would follow a LRU policy.
      - Configurable in most browsers.
    - when a connection lifetime period expires.
      - Configurable in some browsers.
    - when a timeout period expires to read a response from a server.
      - Not configurable in most browsers.
      - c.f. SO_TIMEOUT (Socket.setSoTimeout())
  - A server may want to close a connection when a timeout period expires to read a request from a client.
    - Configurable in most servers.
      - 5 seconds in Apache
    - c.f. SO_TIMEOUT (Socket.setSoTimeout())
      - Sets the timeout period for SocketInputStream.read()
      - ```
        ServerSocket serverSocket = new ServerSocket(…);
        Socket client = serverSocket.accept();
        client.setSoTimeout(30000);     // 30 seconds
        client.getInputStream().read(); // Read a request
        ```

Note: SO_KEEPALIVE

- java.net.SocketOptions.SO_KEEPALIVE
- This "keepalive" is nothing to do with HTTP's persistent connection handling.
- It is "keepalive" for TCP.
  - OS-dependent. 2 hours, usually.
  - Not configurable from Java API
    - Only a getter method is available: Socket.getKeepAlive()
  - Configurable with a kernel parameter.
  - When no data has been exchanged between a client and a server (in either direction) for SO_KEEPALIVE, a prove packet is sent to the peer.
    - If the peer is working normally, nothing occurs.
    - Otherwise, the local socket is closed.

HW 28-2

- Implement persistent HTTP connections
- Keep the original connection-per-resource policy.
  - Use the connection-per-resource policy
    - if a client states HTTP 1.0 without the Connection header field
  - Otherwise, use persistent connections.
    - A client states HTTP 1.1 with/without the Connection header field
    - A client states HTTP 1.0 with the Connection header field
- Set a certain timeout period on Socket.setSoTimeout().
- Have your server state HTTP 1.0 in its responses.
  - A server can state HTTP 1.1 only if it implements all HTTP 1.1 features.
        ```

- Integrate your thread pool with your web server
  - Thread-per-connection with a thread pool
    - if a persistent connection is used.
  - Otherwise, thread-per-request with a thread pool
  - Implement both. Use either of them at runtime depending on whether a persistent conn is used.
    - Implementation suggestion: Use the Strategy design pattern
- Integrate your access counter and caching mechanisms with your web server

- Re-structure (or refactor) your code in an object-oriented fashion.
  - Divide different functionalities, features or responsibilities to different classes and methods.
  - Make your code easy-to-read, easy-to-maintain/extend and bug-resilient.
    - e.g., minimize the # of conditional statements

TinyHttpd

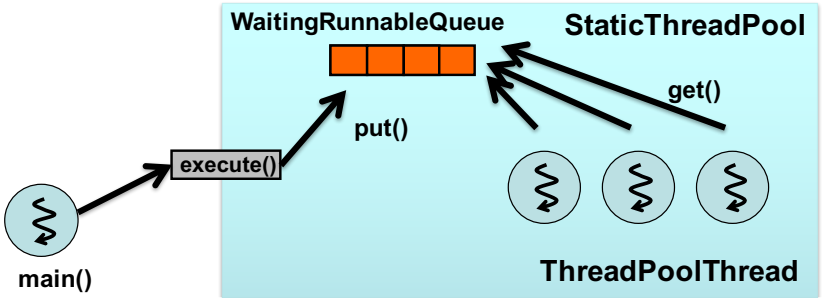
- Authenticator authenticator;
  - References your web server's authenticator
    - An instance of BasicAuthenticator, DigestAuthenticator or NullAuthenticator
- init()
  - Creates an authenticator instance and assigns it to the variable authenticator;
  - Sets up a server-side socket and wait for an incoming request with accept()
- getAuthenticator()
- main()
  - Reads command line arguments
    - Port #
    - -b : basic authentication
    - -d : digest authentication
    - -n : no authentication
    - e.g., java …/…/…/TinyHttpd 8888 -b

HttpExchange

- Represents a pair of a request and a response.
  - Constructor: HttpExchange(Socket, ServerSocket)
    - takes a client socket, reads a request, parses it, and stores all information contained in the request.
  - Representative methods
    - getLocalAddress(), getRemoteAddress(), getRequestCommand(), getRrequestURI(), getProtocolName(), getProtocolVersion(), getRequestHeaders(), getRequestHeader(String key), getRequestBody()…
    - makeSuccessfulResponse(), makeErrorResponse, setResponseHeader(String key, String value), getResponseHeader(String key), getResponseHeaders(), setResponseBody(…), sendResponse()…
  - Go ahead and implement extra methods.

HttpHandler

- HttpHandler implements the Runnable interface
  - Responsible for processing a pair of a request and a response
  - queued to a thread pool
- In TinyHttpd's init()
  - StaticThreadPool pool = new StaticThreadPool(…);
    while(true){
    Socket client = serverSocket.accept();
    pool.execute(new HttpHandler(this, client, serverSocket)); }



- HttpHandler.run()
  - ```
    try{
        HttpExchange ex = new HttpExchange(clientSocket, serverSocket);
      try{
          server.getAuthenticator().authenticateRequest(ex);
          ex.makeSuccessfulResponse(…);
      }catch(AuthenticationFailedException afe){
          ex.makeErrorResponse(afe);
      }
    }catch(BadRequestException bre){
        ex.makeErrorResponse(bre)
    }
    ```

CS681 Individual Project

- Implement "something" extra on top of your web server code base (i.e., HW 28-2 solution)
- Suggestions
  - DB-backed web server
    - Store all HTML files in a SQL or non-SQL DB
      - Convert an incoming HTTP request to a DB query (SQL query)
      - Return a query result as a HTTP response
  - A simple Servlet-like container
    - Receives a HTTP request
      - e.g., http://localhost:8888/tinyservlet/HelloWorld
    - Dynamically instantiates a requested server-side program (e.g., HelloWorld) and call doGet() on it.
    - **`HttpServlet`**
      - `public void doGet(HttpServletRequest request, HttpServletResponse response)`
    - ```
      class HelloWorld extends HttpServlet{
          public void doGet(…){
              response.setContentType("text/html");
              PrintWriter w = response.getWriter();
              w.println("<html>Hello World</html>");
      } }
      ```

- Java Reflection API allows you to instantiate a class by supplying its class name.
  - Instead of calling its constructor
  - Supply "edu.umb.cs.tinyservlet.HelloWorld" to Java Reflection API and instantiate HelloWorld
- Downcast the instance to HttpServlet
- Call doGet() on the instance

– Others
- Implementing WebDAV, iCal, etc.
- Integrating an external authentication service (e.g., Facebook, Google Sign-In)
- Implementing REST APIs
  - Supporting JSON, YAML and other files, not only HTML and image files.

Notes

- You MUST extend your HW 28-2 solution.
- Do not implement trivial (i.e., unsecure) authentication mechanisms.
- A suggestion to develop a Servlet-like container is NOT to use Servlet and develop a Servlet apps.
- Do not copy someone else's code.
  - Every single student who steals code from someone else and who gets code stolen by someone else will earn 0 (ZERO) point.

Outcomes from Your Project

- Submit me
  - Java source code
  - Ant build file
  - Memo/readme that explains how to run your code.
- Deadline: May 20 (Fri) midnight.
  - **FIRM. ABSOLUTELY NO EXTENSIONS.**
- CS680 Deadline: May 13 (Fri) midnight
  - **FIRM. ABSOLUTELY NO EXTENSIONS.**