HyperText Transfer Protocol (HTTP) Review

Carey Williamson

iCORE Chair and Professor

Department of Computer Science

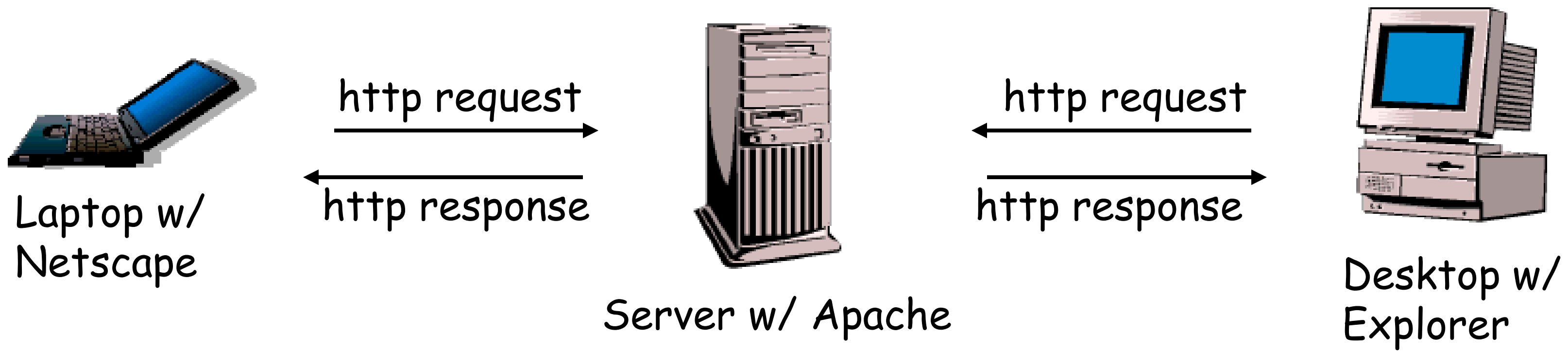
University of Calgary

Slide content courtesy of Erich Nahum (IBM Research), modified by Xifan Zheng,2013,1,22

Today’s tutorial

- Introduction to HTTP
- HTTP request type/format
- HTTP response type/format
- How Web Server works
- Hint for Assignment1

Introduction to HTTP



- HTTP: HyperText Transfer Protocol
  - Communication protocol between clients and servers
  - Application layer protocol for WWW (World Wide Web)
  - Deliver virtually all files and other data: HTML files, image files, query results, or anything else.
  - Through TCP/IP sockets
- Client/Server model:
  - Client: browser that requests, receives, displays object
  - Server: receives requests and responds to them

Introduction to HTTP

- After delivering the response, the server closes the connection (making HTTP a *stateless* protocol)
- Protocol consists of various operations
  - Few for HTTP 1.0 (RFC 1945, 1996)
  - Many more in HTTP 1.1 (RFC 2616, 1999)
  - Faster response, allowing multiple transactions over a single persistent connection.
  - Faster response and great bandwidth savings, by adding cache support.
  - Faster response for dynamically-generated pages, by supporting chunked encoding,
  - Efficient use of IP addresses, multiple domains are served from a single IP address.

HTTP Request Generation

- User clicks on something
- Uniform Resource Locator (URL): scheme+URL. a domain +a port number+ the path of the resource or the program to be run
  - `http://www.cnn.com`
  - `http://www.cpsc.ucalgary.ca`
  - `https://www.paymybills.com`
  - `ftp://ftp.kernel.org`
- Different URL schemes map to different services
- Hostname: a domain name assigned to a host computer. host's local name+its parent domain's name

HTTP Request Generation

- Hostname is converted from a name to a 32-bit IP address (DNS lookup, if needed or local hosts file)
- It is possible for a single host computer to have several hostnames; Any domain name can also be a hostname, If have an IP address

- Connection is established to server (TCP)

What Happens Next?

- Client downloads HTML document
  - Sometimes called "container page"
  - Typically in text format (ASCII)
  - Contains instructions for rendering (e.g., background color, frames)
  - Links to other pages
- Many have embedded objects:
  - Images: GIF, JPG (logos, banner ads)
  - Usually automatically retrieved
    - I.e., without user involvement
    - can control sometimes (e.g. browser options, junkbusters)

```
<html>
<head>
<meta
name="Author"
content="Erich Nahum">
<title> Linux Web
Server Performance
</title>
</head>
<body text="#00000">
<img width=31
height=11
src="ibmlogo.gif">
<img
src="images/new.gif>
<h1>Hi There!</h1>
Here's lots of cool
linux stuff!
<a href="more.html">
Click here</a>
for more!
</body>
</html>
```

sample html file

Web Server Role

- Respond to client requests, typically a browser
  - Can be a proxy, which aggregates client requests (e.g., AOL)
  - Proxy: It receives requests from clients, and forwards those requests to the intended servers. commonly used in firewalls, for LAN-wide caches, or in other situations.
  - Could be search engine spider or robot (e.g., Keynote: Web monitoring service that **checks** Web applications, **notifies** you whenever they become inaccessible, return incorrect data, or respond slowly to connection requests.)
- May have work to do on client's behalf:
  - Is the client's cached copy still good?
  - Is client authorized to get this document?

Web Server Role

- Hundreds or thousands of simultaneous clients
- Hard to predict how many will show up on some day, no sense of how their application infrastructure will handle large increases in traffic (e.g., “flash crowds”, diurnal cycle, global presence)
- Many requests are in progress concurrently

UNIVERSITY OF CALGARY

HTTP Request Format

```
GET /images/penguin.gif HTTP/1.0
User-Agent: Mozilla/0.9.4 (Linux 2.2.19)
Host: www.kernel.org
Accept: text/html, image/gif, image/jpeg
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: B=xh203jfsf; Y=3sdkfjej
<cr><lf>
```

- Messages are in ASCII (human-readable)
- Initial Request Line
- CRLF indicate end of headers,
- Headers may communicate private information
  (browser, OS, cookie information, etc.)

UNIVERSITY OF CALGARY

HTTP Request Format

```
GET /images/penguin.gif HTTP/1.0
User-Agent: Mozilla/0.9.4 (Linux 2.2.19)
Host: www.kernel.org
Accept: text/html, image/gif, image/jpeg
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: B=xh203jfsf; Y=3sdkfjej
<cr><lf>
```

- The User-Agent: header identifies the program that's making the request—Mozilla: Firefox

HTTP Request Types

Called Methods:

- GET: retrieve a file (95% of requests)
- HEAD: just get meta-data (e.g., mod time),
  - ✓ Asks the server to return the response headers only
  - ✓ Check characteristics of a resource without downloading it
  - ✓ Saving bandwidth
- POST: submitting a form to a server.
  - ✓ It's usually a program to handle the data you're sending
  - ✓ HTTP response is normally program output, not a file
- PUT: store enclosed document as URI

HTTP Request Types

- DELETE: removed named resource
- LINK/UNLINK: in 1.0, gone in 1.1
- TRACE: http “echo” for debugging (added in 1.1)
- CONNECT: used by proxies for tunneling (1.1)
- OPTIONS: request for server/proxy options (1.1)

Response Format

- Similar format to requests (i.e., ASCII)

```
HTTP/1.0 200 OK
Server: Tux 2.0
Content-Type: image/gif
Content-Length: 43
Last-Modified: Fri, 15 Apr 1994 02:36:21 GMT
Expires: Wed, 20 Feb 2002 18:54:46 GMT
Date: Mon, 12 Nov 2001 14:29:48 GMT
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Set-Cookie: PA=wefj2we0-jfjf
<cr><lf>
<data follows…>
```

HTTP Response Types

- 1XX: Informational (def'd in 1.0, used in 1.1)

  `100 Continue, 101 Switching Protocols`
- 2XX: Success

  `200 OK, 206 Partial Content`
- 3XX: Redirection

  `301 Moved Permanently, 304 Not Modified`
- 4XX: Client error

  `400 Bad Request, 403 Forbidden, 404 Not Found`
- 5XX: Server error

  `500 Internal Server Error, 503 Service Unavailable, 505 HTTP Version Not Supported`

UNIVERSITY OF CALGARY

Outline of an HTTP Transaction

- This section describes the basics of servicing an HTTP GET request from user space
- Assume a single process running in user space, similar to Apache 1.3
- We'll mention relevant socket operations along the way

```
initialize;
forever do {
   get request;
   process;
   send response;
   log request;
}
```

server in a nutshell

Readying a Server

```
s = socket(); /* allocate listen socket */
bind(s, 80);  /* bind to TCP port 80    */
listen(s);    /* indicate willingness to accept */
while (1) {
    newconn = accept(s); /* accept new connection */
```

- First thing a server does is notify the OS it is interested in WWW server requests; these are typically on TCP port 80. Other services use different ports (e.g., SSL is on 443)
- Allocate a socket and bind()'s it to the address (port 80)
- Server calls listen() on the socket to indicate willingness to receive requests
- Calls accept() to wait for a request to come in (and blocks)
- When the accept() returns, we have a new socket which represents a new connection to a client

Processing a Request (1 of 2)

```
remoteIP = getsockname(newconn);
remoteHost = gethostbyname(remoteIP);
gettimeofday(currentTime);
read(newconn, reqBuffer, sizeof(reqBuffer));
reqInfo = serverParse(reqBuffer);
```

- getsockname() called to get the remote host name
  - for logging purposes (optional, but done by most)
- gethostbyname() called to get name of other end
  - again for logging purposes
- gettimeofday() is called to get time of request
  - both for Date header and for logging
- read() is called on new socket to retrieve request
- request is determined by parsing the data
  - Example: “GET /images/jul4/flag.gif”

Processing a Request (2 of 2)

```
fileName = parseOutFileName(requestBuffer);
fileAttr = stat(fileName);
serverCheckFileStuff(fileName, fileAttr);
open(fileName);
```

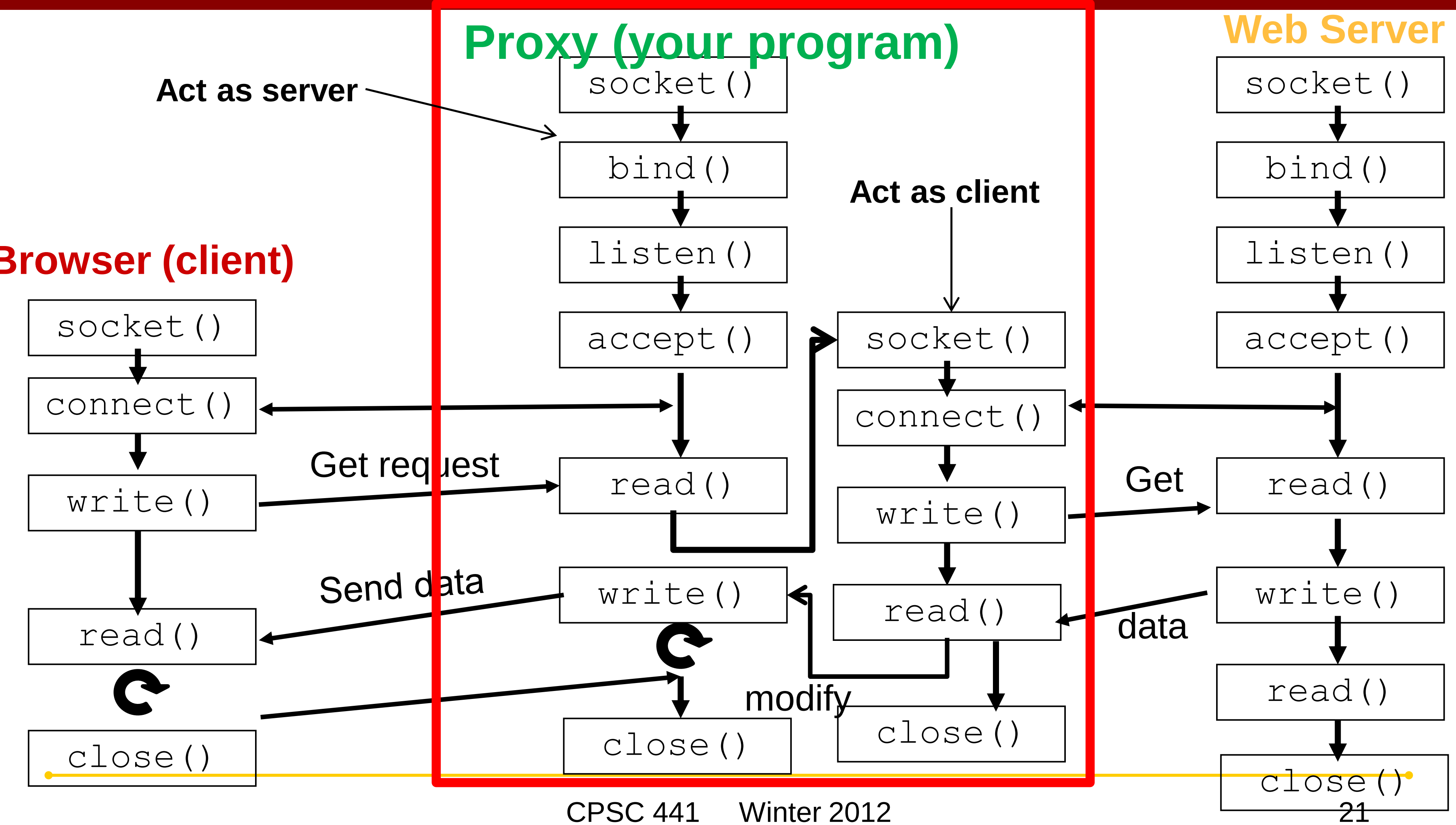
- stat() called to test file path
  - to see if file exists/is accessible
  - may not be there, may only be available to certain people
  - "/microsoft/top-secret/plans-for-world-domination.html"
- stat() also used for file meta-data
  - e.g., size of file, last modified time
  - "Has file changed since last time I checked?“
- might have to stat() multiple files and directories
- assuming all is OK, open() called to open the file

Responding to a Request

```
read(fileName, fileBuffer);
headerBuffer = serverFigureHeaders(fileName, reqInfo);
write(newSock, headerBuffer);
write(newSock, fileBuffer);
close(newSock);
close(fileName);
write(logFile, requestInfo);
```

- read() called to read the file into user space
- write() is called to send HTTP headers on socket
  (early servers called write() for *each header!)*
- write() is called to write the file on the socket
- close() is called to close the socket
- close() is called to close the open file descriptor
- write() is called on the log file

Hint for Assignment1



UNIVERSITY OF CALGARY

Thanks for attending!!