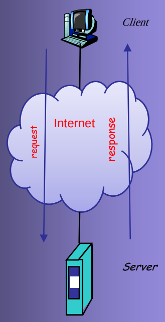


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 8.1: HTTP (Part 1)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

- Part 1:
  - Learn the essentials of the two main versions of HTTP protocol
    - HTTP 1.0 [RFC 1945]: http://www.ietf.org/rfc/rfc1945.txt
    - HTTP 1.1 [RFC 2616]: http://www.ietf.org/rfc/rfc2616.txt
- Part 2:
  - Learn how to write HTTP Clients using WebClient, WebRequest, WebResponse, etc.
  - Learn how to request resources requiring authentication
  - Learn how to pass HTTP request through a proxy server
  - Learn how to write a simple HTTP server

Introduction to Internet Protocols

- Internet protocols evolve through a community discussion process called Request for Comments (RFC)
  - All members of the Internet community (including network designers, operators, vendors, researchers, etc) can participate.
- Discussions are supervised by the Internet Engineering Task Force (IETF): http://www.ietf.org
  - Actual technical work is done by working groups, organized by topics in several areas.
  - IETF is an open international community of individuals and organizations interested in the evolution of the Internet.
- The details of each internet protocol are documented and numbered with prefix RFC and is made available online
  - Check this link for details: http://www.ietf.org/rfc.html

HTTP

- HTTP stands for HyperText Transfer Protocol
- An application protocol based on client-server architecture, designed for delivering hypermedia information on the web
- The design goals of HTTP are:
  - light protocol: not consuming too much resources
  - fast protocol: need to retrieve many widely distributed documents as fast as possible
- According to the RFCs, HTTP is independent of the transport layer protocol
  - But, in practice HTTP servers use TCP on the default port# 80
- The first version was given version # 0.9. But, the two prevalent versions are 1.0 and 1.1.
  - For more details, refer to: http://www.ietf.org/rfc/rfc1945.txt (HTTP 1.0) and http://www.ietf.org/rfc/rfc2616.txt (HTTP 1.1).

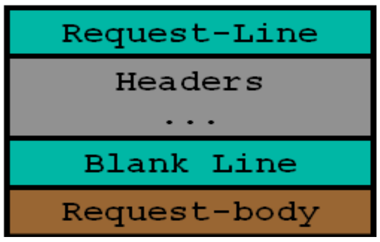
HTTP 1.0

- HTTP 1.0 is a stateless protocol designed to a allow a client to make a request to the server and get a response.
  - Stateless means servers retain no information about past requests.
- The protocol specifies the format of the request and the format of the response among other things.
- Interaction between client and server has 4 phases:
  - client connects to server
  - client sends request to server
  - server sends response to client
  - server closes connection

HTTP 1.0 …

Format of HTTP 1.0 Request:

- A request has four parts: The request line, optional headers (0 or more), blank line, and optional body.
- The Request Line and each header line must end with CRLF (“\r\n”).
- The Request body is sent in binary format.



HTTP 1.0 …

- The request line has three parts and terminates with CRLF:
  Method Request-URI HTTP-Version CRLF
- Methods (case sensitive) include:
  – **GET**: request document named by request-URI. Additional parameters for the request are appended to the request-URI
  – **HEAD**: request only header information about request-URI
  – **POST**: similar to GET but request parameters are provided through the Message Body.
- Request-URI specifies the full path of the resource being requested and must begin with “/”:
  e.g: /swe344/lectures/lecture1.html
  – URI stands for Universal Resource Identifier (superset of URL and URN)
- HTTP-version specifies the version of HTTP used by the client making the request.
  – Values are: HTTP/1.0 or HTTP/1.1

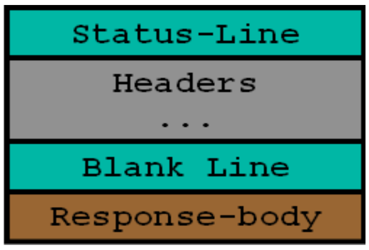
HTTP 1.0 …

- Headers are used by both the client making a request or by the server responding to the request.
  – Headers provide information about the request, response, object being requested/sent, server or client.
- The headers have the form "Header-Name: value", ending with CRLF.
  – The header name is not case-sensitive (but the value may be).
  – Any number of spaces or tabs may be between the ":" and the value.
- Some common header names:
  – Accept: what format is acceptable (client)
  – Content-Length: length of the message (client/server)
  – Content-Type: type of the content (server)
  – Date: date sent (sever)
  – Expires: expiry date of the content (server)
  – Last-Modified: Last modification date (server)

HTTP 1.0 …

Format of HTTP 1.0 Response:

- A response also consists of four parts: The status line, optional headers (0 or more), blank line, and optional response body.
- The Response line and each header line must end with CRLF (“\r\n”).
- The response body is sent in binary format.



HTTP 1.0 …

- The status line has three parts, terminated by CRLF:

  HTTP-Version Status-Code Status-Text CRLF
- HTTP-Version specifies the HTTP of the server responding to the request: HTTP/1.0 or HTTP/1.1
- Status-code is a three-digit integer indicating the status of the request, Status-Text explains the status-code.
  - First digit identifies the category of the response
    - **1xx** indicates an informational message only
    - **2xx** indicates success of some kind
    - **3xx** redirects the client to another URL
    - **4xx** indicates an error on the client's part
    - **5xx** indicates an error on the server's part
  - The most common status codes are:
    - **200 OK** : The request is successful
    - **404 Not Found** : The requested resource doesn't exist.
    - **500 Server Error** : An unexpected server error.

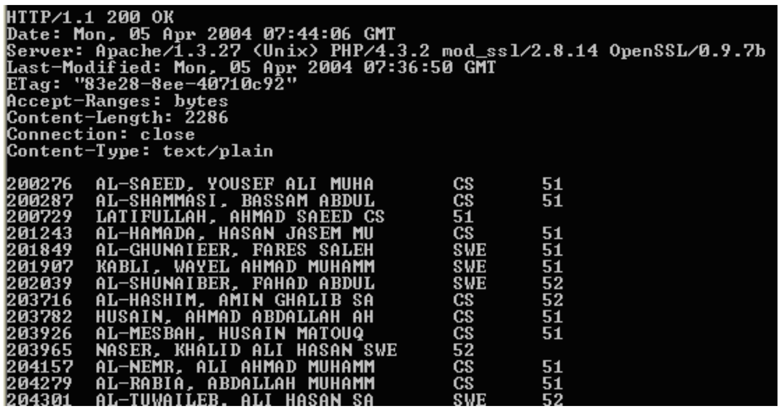
Using Telnet

- TELNET can be used to test an HTTP server:

  telnet www.ccse.kfupm.edu.sa 80

  GET /~bmghandi/swe344/swe344.txt HTTP/1.0 ><Enter>

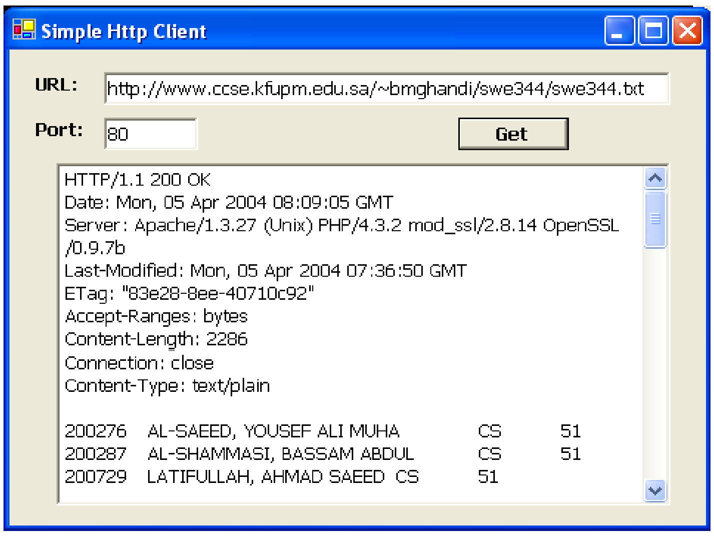
  <blank Line> <Enter>



```
HTTP/1.1 200 OK
Date: Mon, 05 Apr 2004 07:44:06 GMT
Server: Apache/1.3.27 (Unix) PHP/4.3.2 mod_ssl/2.8.14 OpenSSL/0.9.7b
Last-Modified: Mon, 05 Apr 2004 07:36:50 GMT
ETag: "83e28-8ee-40710c92"
Accept-Ranges: bytes
Content-Length: 2286
Connection: close
Content-Type: text/plain

200276  AL-SAEED, YOUSEF ALI MUHA       CS      51
200287  AL-SHAMMASI, BASSAM ABDUL       CS      51
200729  LATIFULLAH, AHMAD SAEED CS      51
201243  AL-HAMADA, HASAN JASEM MU       CS      51
201849  AL-GHUNAIEER, FARES SALEH       SWE     51
201907  KABLI, WAYEL AHMAD MUHAMM       SWE     51
202039  AL-SHUNAIBER, FAHAD ABDUL       SWE     52
203716  AL-HASHIM, AMIN GHALIB SA       CS      52
203782  HUSAIN, AHMAD ABDALLAH AH       CS      51
203926  AL-MESBAH, HUSAIN MATOUQ        CS      51
203965  NASER, KHALID ALI HASAN SWE     52
204157  AL-NEMR, ALI AHMAD MUHAMM       CS      51
204279  AL-RABIA, ABDALLAH MUHAMM       CS      51
204301  AL-TUWAILEB, ALI HASAN SA       SWE     52
```

Develop a Web Client



URL: http://www.ccse.kfupm.edu.sa/~bmghandi/swe344/swe344.txt

Port: 80

```
HTTP/1.1 200 OK
Date: Mon, 05 Apr 2004 08:09:05 GMT
Server: Apache/1.3.27 (Unix) PHP/4.3.2 mod_ssl/2.8.14 OpenSSL
/0.9.7b
Last-Modified: Mon, 05 Apr 2004 07:36:50 GMT
ETag: "83e28-8ee-40710c92"
Accept-Ranges: bytes
Content-Length: 2286
Connection: close
Content-Type: text/plain

200276  AL-SAEED, YOUSEF ALI MUHA       CS      51
200287  AL-SHAMMASI, BASSAM ABDUL       CS      51
200729  LATIFULLAH, AHMAD SAEED  CS     51
```

Develop a Web Client (cont.)

```
void OnGetClicked(object sender, System.EventArgs e) {
  String url = urlBox.Text;
  int doubleSlahIndex = url.IndexOf("//");
  if (doubleSlahIndex>0) {  //remove protocol part
    doubleSlahIndex+=2;
    url = url.Substring(doubleSlahIndex);
  }
  int pathIndex = url.IndexOf('/');
  string host = url.Substring(0, pathIndex);
  string path = url.Substring(pathIndex);
  int port = int.Parse(portBox.Text);
  client = new TcpClient(host, port);
  stream = client.GetStream();
  reader = new StreamReader(stream);
  writer = new StreamWriter(stream);
  String command = "GET "+path+ " HTTP/1.0"+"\r\n";
  writer.WriteLine(command); //WriteLine will add the second "\r\n"
  writer.Flush();
  string input;
  while((input = reader.ReadLine()) != null)
    resultBox.Text += input + "\r\n";
}
```

HTTP 1.1

- HTTP 1.1 has been defined to address new needs and overcome shortcomings of HTTP 1.0
- Improvements include:
  - Faster response, by allowing multiple transactions to take place 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***, which allows a response to be sent before its total length is known.
  - Efficient use of IP addresses, by allowing ***multiple domains*** to be served from a single IP address.
- To achieve these improvements, HTTP 1.1 added extra specifications on both the client and the server

HTTP 1.1 : Client Specifications

Client must send Host: header with each request

- In HTTP 1.1, a server at one IP address can be *multi-homed.*
  - i.e. the home of several websites.
  - e.g. both, "www.site1.com" and "www.site2.com" can be hosted on the same server.
- Several domains living on the same server is like several people sharing one telephone.
  - A caller knows who he is calling, but the receiver of the call doesn't.
- HTTP 1.1 requires a client to specify a host name (and port) that the request is intended for, using the **Host:** header.

```
GET /path/file.html HTTP/1.1
Host: www.host1.com:80
[blank line here]
```

  - Note: ":80" isn't required in the above, since it is the default HTTP port

HTTP 1.1 : Client Specifications …

Client must be able to handle chunked response

- A HTTP 1.1 server may start sending a response before knowing its total length – for dynamically generated content.
- The response will include the Transfer-Encoding: header instead of Content-Length: header.
- Each chuck is preceded by its length (in hex) in the response body. Length value of 0 indicates last chuck.

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Transfer-Encoding: chunked

1a
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
```

The length of the text data is 42 bytes (1a + 10, in hex).

The data itself is: **abcdefghijklmnopqrstuvwxyz1234567890abcdef**

HTTP 1.1 : Client Specifications …

Client should be able to handle Persistent Connection

- In HTTP 1.0, connection is closed after processing request
  - Opening and closing TCP connections takes a substantial amount of CPU time, bandwidth, and memory.
- Persistent connection is the default in HTTP 1.1.
  - After establishing connection, the client send several requests in series (called *pipelining*), and read the responses in the same order as the requests.
  - If a client cannot handle Persistent Connection, it must send the "Connection: Close" header with each request.
  - A server might close the connection before all responses are sent, so a client must keep track of requests and resend them as needed.
  - Client should not pipeline at all if the server won't support persistent connections (if it uses HTTP 1.0, based on a previous response)

HTTP 1.1 : Client Specifications …

Client must be able to handle 100 Continue response

- When a HTTP 1.1 client sends a request to a server, the server might respond with an interim "**100 Continue**" response.
  - It means the server has received the request and has not yet rejected it, the client can continue sending the rest of the request.
  - The purpose is to allow a client sending a request that has request body (using POST or PUT methods) to determine if the server is willing to accept the request (based on the request headers) before sending the request body.
  - If a client will wait for a 100-Continue response before sending the request body, it must send an "Expect: 100-Contine" request-header.
  - A client must not send "Expect: 100-Continue" request-header field if it does not intend to send a request body.
- HTTP 1.1 clients must handle the "100 Continue" response correctly – by sending the message body or by ignoring it if the body is already sent.
- The "**100 Continue**" response is structured like any HTTP response - a status line, optional headers, and a blank line.

```
HTTP/1.1 100 Continue
[blank line here]
```

- Unlike other responses, "100 Continue" response is always followed by another complete, final response.

HTTP 1.1 : Server Specifications

Server must require the Host: header

- Servers must NOT accept HTTP 1.1 requests without the **Host:** header. Instead, it must return a "**400 Bad Request**".

```
HTTP/1.1 400 Bad Request
Content-Type: text/html
Content-Length: 111

<html><body>
<h2>No Host: header received</h2>
HTTP 1.1 requests must include the Host: header.
</body></html>
```

Note: Request without Host: header is acceptable from 1.0 clients.

Server must accept request with absolute URL

- Future versions of HTTP clients will send absolute URL:

```
GET http://www.host.com/path/file.html HTTP/1.2
```

- To enable this protocol transition, HTTP 1.1 servers must also accept this form of request.

HTTP 1.1 : Server Specifications ...

Server must be able to handle chunked request

- Just as HTTP 1.1 clients must accept chunked responses, servers must also accept chunked requests
  - Servers aren't required to generate chunked responses; they just have to be able to receive chunched requests.

Server should support persistent connection.

- Server should support persistent connection or send "Connection: Close" header with each response
  - Must respond to multiple requests through a persistent connection in the same order as the requests.
  - If a request includes the "**Connection: close**" header, the server should close the connection after responding.
  - Server should close an idle connection after some timeout period.

HTTP 1.1 : Server Specifications …

Server must send the "100 Continue" Response

- Upon receiving a request that includes "Expect: 100-Continue", the server MUST either respond with "100 Continue" status and continue to read from the input stream, or respond with a final status code (e.g. "417 Expectation Failed".
- The server MUST NOT wait for the request body before sending the "100 Continue" response.
- A server SHOULD NOT send a "100 Continue" response if the request message does not include an "Expect: 100-Continue" request-header
- Server must follow the "100 Continue" response with the final response, once the request has been processed.

Server must send the Date: header

- Caching is an important improvement in HTTP 1.1, and can't work without timestamp on responses.
- The **Date:** header contains the current time, in the format
  Date: Fri, 31 Dec 2004 23:59:59 GMT
- All responses except those with 100-level status (but including error responses) must include the **Date:** header.

HTTP 1.1 : Server Specifications …

Server must handle If-Modified-Since:, If-Unmodified-Since:

- These headers are used to support caching.
  - Clients aren't required to use them, but HTTP 1.1 servers are required to honor requests that use them.
- The **If-Modified-Since:** header means if the resource has been modified since the given date, return it as normal, otherwise, return "**304 Not Modified**" response.

```
HTTP/1.1 304 Not Modified
Date: Fri, 31 Dec 1999 23:59:59 GMT
[blank line here]
```

- The **If-Unmodified-Since:** header means return the resource only if it has *not* been modified since the given date, otherwise, return a "**412 Precondition Failed**":

```
HTTP/1.1 412 Precondition Failed
[blank line here]
```

HTTP 1.1 : Server Specifications …

Server must support GET and HEAD methods

- To comply with HTTP 1.1, a server must support at least the GET and HEAD methods.
- If a client requests a method that is not supported, the server should respond with "**501 Not Implemented**".

Server must be backward compatible

- To be compatible with older clients, a HTTP 1.1 servers must support HTTP 1.0 requests.
- In particular, when a request line indicates HTTP 1.0 :
  - don't require the **Host:** header, and
  - don't send the "**100 Continue**" response.

Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- HTTP 1.0 http://www.ietf.org/rfc/rfc1945.txt
- HTTP 1.1 http://www.ietf.org/rfc/rfc2616.txt
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm