Unit 2

Web servers and HTTP
Course team
Developer: Jenny Lim, Consultant
Designer: Chris Baker, OUHK
Coordinator: Dr Li Tak Sing, OUHK
Member: Dr Andrew Lui Kwok Fai, OUHK

External Course Assessor
Prof. Mingshu Li, Institute of Software, Chinese Academy of Sciences

Production
ETPU Publishing Team

Copyright © The Open University of Hong Kong, 2004.
All rights reserved.
No part of this material may be reproduced in any form by any means without permission in writing from the President, The Open University of Hong Kong.

The Open University of Hong Kong
30 Good Shepherd Street
Ho Man Tin, Kowloon
Hong Kong
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>1</td>
</tr>
<tr>
<td>Objectives</td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td><strong>Web server components</strong></td>
<td>4</td>
</tr>
<tr>
<td>Computing platform</td>
<td>4</td>
</tr>
<tr>
<td>Web server software</td>
<td>5</td>
</tr>
<tr>
<td>Document</td>
<td>9</td>
</tr>
<tr>
<td><strong>Document organization</strong></td>
<td>15</td>
</tr>
<tr>
<td>Document trees</td>
<td>15</td>
</tr>
<tr>
<td><strong>Web server process models</strong></td>
<td>21</td>
</tr>
<tr>
<td>Advantages and disadvantages of different process models</td>
<td>23</td>
</tr>
<tr>
<td>The Web browser: some hidden details</td>
<td>24</td>
</tr>
<tr>
<td><strong>HTTP — a closer look</strong></td>
<td>28</td>
</tr>
<tr>
<td>HTTP requests</td>
<td>30</td>
</tr>
<tr>
<td>HTTP responses</td>
<td>30</td>
</tr>
<tr>
<td><strong>Other HTTP features</strong></td>
<td>35</td>
</tr>
<tr>
<td>Virtual hosts</td>
<td>35</td>
</tr>
<tr>
<td>Content negotiation</td>
<td>36</td>
</tr>
<tr>
<td>Persistent connections</td>
<td>39</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>42</td>
</tr>
<tr>
<td>Feedback to activities</td>
<td>43</td>
</tr>
<tr>
<td><strong>Suggested answers to self-tests</strong></td>
<td>45</td>
</tr>
<tr>
<td>References</td>
<td>50</td>
</tr>
<tr>
<td>Readings</td>
<td></td>
</tr>
</tbody>
</table>
Overview

This unit examines the organization of the Web server document tree and the internal structure and process model of the Web server software. The step-by-step mechanism in which a Web browser ‘talks’ to a Web server according to the HyperText Transfer Protocol (HTTP) is described and traced. We will examine the HTTP messages that are exchanged between them, and study the structure and meaning of these messages.

To learn about these concepts you will go through selected online and print materials, as well as sections of the official HTTP protocol specification. You will compare and contrast the protocol specification with an implementation of that specification: your local Apache Web server. You will also conduct a series of experiments with your local Web server to demonstrate how the HTTP communication protocol works.

This unit should take you about four weeks (or about 30 hours of study) to complete. Please make sure that you plan your time carefully.
Objectives

1. *Explain* how Web documents can be stored and organized using a variety of document tree structures on the server.

2. *Understand* how Web servers can operate using different process models; and *compare* and *contrast* the different approaches.

3. *Examine* the structure and meaning of the HTTP messages exchanged between the browser and the server.

4. *Discuss* the important characteristics and features of HTTP.
Introduction

The Web is a virtual information space that never sleeps, and Web servers are responsible for keeping it that way — 24 hours a day, seven days a week, 365 days a year. There were almost 45 million Web servers on the Web as of November 2003 (Netcraft 2003). No matter which way you look at it, this is a phenomenal increase from the 18,957 Web servers that were accounted for in Netcraft’s first survey, dated August 1995 to August 1996 (Netcraft 1995).

Web servers open the doors to the enormous store of documents and executable applications that information providers are willing to provide on the other end. When we talk about a Web server, we may actually mean different things. We could be talking about the hardware and operating system that the Web server runs on, or the Web server software itself, or the documents stored on it. In truth, the Web server is all of these things combined.

Web servers, along with Web browsers, are designed to implement the HyperText Transfer Protocol (HTTP). This allows them to understand incoming requests for documents and send back valid responses. However, Web servers must also know how to process these requests quickly and efficiently, or there’ll be a lot of unhappy users! Modern Web servers now process multiple requests in parallel, rather than one at a time.

The Web was a very different place when HTTP/1.0 was first created in 1991. New features have been proposed and added to HTTP/1.1 that aim to address the limitations of HTTP/1.0. Aside from examining the structure and meaning of HTTP messages, we shall also look at some of the additional features which are aimed at improving the performance and scalability of HTTP.
Web server components

Every request from a user sitting at a Web browser is ultimately served by a Web server at the other end. Web servers are the workhorses of the Web, and just as the Web is composed of many components, a Web server can also be broken down into many parts. The following formula describes what makes up a Web server:

\[
\text{Web server} = \text{computing platform} + \text{Web server software} + \text{documents}
\]

In a nutshell, a Web server is a computer that is connected to the Internet, usually with its own unique IP address. It runs Web server software which responds to requests for resources from Web clients. The requested resources can be static or dynamically generated Web documents which are stored, organized and served from the server’s file system.

In this section, we first describe the components of a Web server, and then explain how these components are used to fulfill its functions.

Computing platform

The computing platform refers to the underlying hardware or software for a system. The platform provides the environment in which a Web server operates and the vital services which it needs. This includes the following:

1. Operating system (O/S) software — handles communication between the hardware and the software applications running on it; also controls the allocation of system resources. The networking software (e.g., TCP/IP stack) and file system software is often built into the O/S, too. Multitasking operating systems, such as Windows NT or UNIX, are generally better suited to running a Web server.

2. Computer hardware — includes the Central Processing Unit (CPU), hard disks, disk controllers, main memory, and network interface card.
3 Network connection — interfaces the Web server to other machines on a network (e.g., the Internet or a local area network). Sufficient communications bandwidth must also be available, as this is likely to be a limiting factor in the Web server’s performance. Bandwidth determines the amount of data that can be carried by the network cable in a given amount of time, and is typically measured in megabytes per second (mbps).

Now that you have briefly examined the computing platform (network, operating system, and file system) let’s move on and examine the main player in our show: the Web software. The Web software is an implementation of a standard communication protocol, the HyperText Transfer Protocol (HTTP). Both the Web browser and the Web server software implement pieces of this communication protocol so that they can ‘talk’ to each other.

**Web server software**

A Web server performs a simple and straightforward task. It continually listens for HTTP requests coming from Web clients, usually over a network connection. When an incoming request is received, the server interprets the request and tries to find the resource that is requested by the user. In its simplest form, the resource could be an HTML page residing on a local disk, but it can also be many other things, such as an image file, Java applet, or a script which generates a webpage in real-time. All resources can be uniquely identified via a Uniform Resource Indicator (URI).

![Figure 2.2](image)

**Figure 2.2** How a Web server responds to requests for resources

Assuming the server is able to locate the requested resource, it retrieves the file from disk and sends it back to the Web browser over the network. In case of errors (e.g., the file is not found), the server will respond with an explanation. Compared with the Web browser, Web servers have a simple structure. They merely deliver data without knowing what it is about. They only see a stream of bytes coming in and going out. Browsers have a more complex task since they have to determine what type of data has been received and how it should be rendered to the user.
There are other tasks that Web servers can handle. They can log all the activity that takes place: such as the IP address, date/time and request made for each client connection. They can be configured to authenticate users before allowing them access to certain files or directories. For requests which go beyond merely retrieving files from disk, Web servers can also interface with application gateways such as relational databases, executable programs or email servers for further processing.

Criteria for choosing Web server software

What are the criteria for choosing Web server software? Depending on your situation, any or all of the following might need to be considered:

- compatibility with your current operating environment and hardware platform;
- security features;
- ease of use;
- vendor and technical support;
- availability of upgrades; and
- good performance reviews.

Surprisingly, cost is not such a big factor since Web server software can often be downloaded at no charge or acquired at minimal cost. It may even come with your operating system for free!

In the next activity, you’ll view some resources which could be helpful when picking out a Web server software and platform.

Activity 2.1

1 Since August 1995, Netcraft, a British Web and network consultancy, has conducted a monthly survey of Web server software usage on Internet-connected computers. Go to the following website to find out which Web server software programs are most widely used:


2 Find out which operating systems and Web server software others are using through this online tool:

   http://uptime.netcraft.com/up/graph


(Note: You’ll be surprised to know what Microsoft is using!)
Consult this checklist of the essential features and capabilities you should look for in a Web server:

http://slis-two.lis.fsu.edu/~planning/server/features.htm

Establishing a connection between an HTTP client and server

How does a Web server use the network port for processing HTTP transactions? As you’ve seen in Unit 1, computers communicate over the network by establishing a connection with the port, or access point, on the server they want to reach. The port is an identifying number which is assigned to a service running on a server, analogous to a phone extension. The Web server only needs to know how to read requests from and write requests to the World Wide Web service port (port 80 by default). The server does not need to understand how the network connection is started and managed. This is all taken care of by lower-level protocols such as TCP/IP.

Here are the steps for establishing a connection between an HTTP client and server (using HTTP/1.1):

1. The client must locate the server.
2. The client’s system software initiates TCP handshake by sending a packet to the server, requesting a connection.
3. The server’s system software sends a packet back to the client, agreeing to set up a connection.
4. The client program is connected to the new network connection.
5. The server program is connected to the new network connection.
6. The client sends multiple HTTP requests through a single connection.
7. The server sends HTTP responses back in the same order as the requests.
8. The server closes the connection.

The steps above are summarized in figure 2.3.
The next activity illustrates how Web servers and browsers communicate via network ports.

In the following activity, you will start a Web server which ‘listens’ for requests on a different TCP/IP port from the default HTTP port (80). Upon successfully completing this exercise you will have two Web servers running on your local computing platform, each Web server listening on a separate port.

**Activity 2.2**

1. Start your localhost Web server which is configured to listen to port 80 (as specified in `httpd.conf`).

2. Create a new version of the configuration file (`httpd.conf`) so that the server now listens to port 8080 and records its process ID to a file called `httpd8080.pid`. Change the following directives and save the new configuration file as `httpd8080.conf` in the `logs` directory:

   ```
   Listen 8080
   PidFile logs/httpd8080.pid
   ```

3. On the command line, start another server using this new configuration by issuing the command:

   ```bash
   % httpd -f xxxx/httpd8080.conf
   ```

   where `xxxx` is the path where `httpd8080.conf` can be found.
4 Retrieve a Web document on the Web server listening to port 8080:

   http://localhost:8080/index.html

Note how the URL is formed by attaching the port number to the host name, using a colon to separate the two.

5 Retrieve the same Web document on the Web server listening to port 80:

   http://localhost/index.html

The default port 80 is assumed since no port number was explicitly provided in the URL.

6 Use the `netstat` command and observe that there are two Apache Web server processes running. This proves that even though you viewed the same homepage twice, it was served to you by two different Web servers, depending on the port number used.

7 Terminate or ‘kill’ the Web server on port 8080 using

   `% kill PID`

   where PID is the process ID of the Web server as recorded in `logs/httpd8080.pid`.

   You may also terminate the Web server on port 80 in a similar manner. The process ID for this Web server will be in `logs/httpd.pid`.

   **Note:** Setting up two Web servers listening to different ports can be even more useful when each server serves different sets of documents.

   For example, the employees of ABC Books may be given access to a different website from the ordinary public. Employees access this special website at `http://www.abcbooks.com.hk:8080`, while their customers continue to access the public website at `http://www.abcbooks.com.hk`.

   You will learn how to assign different sets of documents to Web servers in Activity 2.4.

---

**Document**

In the previous section, you learned that the Web server’s main job is sending documents over the network. Without documents to serve there would be no reason for the Web server to exist! From the beginning, Tim Berners-Lee envisioned the Web as a delivery platform for multimedia documents, not just text-based documents. Here are some common media types that are handled over the Web:

- images;
- audio;
- video;
• slide presentations; and
• spreadsheets;

How does the Web manage to deliver and render such a wide variety of content? In this section, you will learn how the Web has been designed to recognize and serve all the different types of information people might want to put on it.

**Mime types and file extensions**

Documents are stored on a Web server in the form of binary encoded data files. The bytes within a file can be organized in a variety of ways, depending on the nature of its contents and the software that will read and use the data. For example, image files can be encoded in the GIF, JPEG or PNG format, while video files can be encoded in Quicktime or RealMedia format. Another term for the way data is represented and structured in a file is **encoding**.

The Web server software doesn’t need to know the structure of the bytes within the documents that it sends over the network. However, the server does need to send metainformation along with the transmitted file which will allow the Web browser to recognize the media type of the incoming byte stream and decide how to process it, for example, whether to display the bytes as an image or play them as an audio stream.

You’ve already seen how Web browsers are able to render different kinds of media using a type system called **Multipurpose Internet Mail Extensions (MIME)**. Now we’ll examine how the Web server also uses mime types to tell the browser what kind of data is being sent within an HTTP message.

Web authors and information providers follow a long-standing convention when assigning names to the files they create and serve. The file extension used in a filename is also used to reflect the nature of its contents. For example, a file containing an HTML document will be called **something.html**, an image in GIF format will be called **something.gif** and a WORD document will be called **something.doc**.

The Web server is configured with a list of recognized file extensions and their corresponding MIME-types. When a document is sent out, the server maps its file extension to its MIME-type, and then puts this information in a **Content-Type** header within the HTTP response message.

Aside from the **Content-Type** header, there are other headers used by the server to give the browser more information about the transmitted file and its contents. The following table lists some of these headers.
Table 2.1 HTTP entity headers, based on the HTTP specification (RFC2616)

<table>
<thead>
<tr>
<th>Entity headers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type</td>
<td>Media type of the data sent to the recipient. The example below shows the Content-Type for a file with the .html extension.</td>
</tr>
<tr>
<td></td>
<td>Content-Type: text/html; charset=ISO-8859-4</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>When present, its value indicates whether the message has been transformed, usually via a compression method. This also lets the recipient know how the data should be decoded.</td>
</tr>
<tr>
<td></td>
<td>Content-Type: application/postscript</td>
</tr>
<tr>
<td></td>
<td>Content-Encoding: gzip</td>
</tr>
<tr>
<td>Content-Length</td>
<td>This is the size of the entity-body, in decimal number of OCTETs.</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 3495</td>
</tr>
</tbody>
</table>

Here’s how the MIME header fields might appear within an HTTP response:

   Content-Type: text/html
   Content-Length: 2294

The MIME-type system offers great flexibility. Web servers and Web browsers can easily extend the MIME specification to add new MIME types. All you need to do is make sure that the server is configured to output the new type and that browsers are configured to expect the new type.

Now let’s see how our Web server and Web browser use MIME types and file extensions to serve Adobe Portable Document Format (PDF) files.

Activity 2.3

1 The Adobe Acrobat Reader is an external viewer or application for the Portable Document Format (PDF). The Web browser cannot read PDF files — it needs a helper program, the Adobe Acrobat PDF reader. If you do not have the Adobe Acrobat Reader installed on your computer, download it from http://www.adobe.com and install it according to the directions on the website.

2 Some MIME types are registered with IETF in the Internet media type registry, a list of widely used MIME type/file extension pairs. Read this file to see how application/pdf has been registered as an official Internet MIME-type name for the Adobe Portable Document Format:

   http://www.iana.org/assignments/media-types/application/
Explore http://www.iana.org/assignments/media-types/ so that you can identify other common Internet registered MIME types.

3 Open your Web server’s list of MIME types in 
$(Apache_rootdir)/conf/mime.types and look for application/pdf. You should see something like this:

```
# MIME typeExtension
application/pdf pdf
```

When the Web server sees a file with a .pdf file extension it sends application/pdf as the MIME type to the Web browser.

4 Ensure that your Web browser is configured to receive and understand the PDF MIME-type and file extension. Refer to Activity 1.3 in Unit 1 if you’re unsure how to do this.

5 Download the latest ABC Books sales brochure (abc_home.pdf) from the course website. Store this document in the root directory of your Web server. Now retrieve this document by typing this URL in your browser:

```
http://localhost/abc_home.pdf
```

Your local Web server is sending the Web browser the application/pdf MIME type. Based on this information, your browser will invoke the Adobe Acrobat Reader in order to display the PDF document.

6 The Web browser and Web server negotiate the MIME Content-Type application/pdf during the transfer of a Web document. To demonstrate this process we will remove the definition of the application/pdf MIME type from the Web server’s configuration and see how the Web browser displays the document abc_home.pdf.

- Kill your Web server.

  Remove the definition of application/pdf by placing a # (the symbol for a comment in the configuration file) in front of the definition for application/pdf in $(Apache_rootdir)/conf/mime.types.

  The change in mime.types should look like this:

  ```
  # MIME typeExtension
  # application/pdf pdf
  ```


- What do you see? The Web server does not send application/pdf as the MIME type to the Web browser because it no longer recognizes this file extension/MIME type combination. Without any additional information about the file
received, the Web browser displays abc_home.pdf according to its 'default' display mechanism — as an ASCII text document.

• Now restore your $mime.types$ file to its original form. Kill your Web server, edit $mime.types$ and remove the #, and then restart your Web server. Check that the browser now displays the PDF document correctly.

**Important note:**

You may not always get the expected results during the exercises with your local Web server (127.0.0.1) and Web browser. For example, after restoring the $mime.types$ file to its original condition, your browser should display the PDF document within the Adobe Acrobat Reader again. If this still doesn’t work correctly, you are experiencing the effects of your Web browser’s cache.

**Caches**

A cache is a set of Web documents stored on the Web client’s local disk. Fetching documents from a cache is a lot faster than fetching them from a remote Web server. Caches, as we shall see in *Unit 6*, are useful for improving Web server performance and response times.

The behaviour of caches can interfere with our experiments and appear to cause errors. Sometimes, when you request a document, you may see unexpected or ‘old’ version of the document. If this is the case you are probably viewing an out-of-date version of the document from your browser’s local cache.

Unfortunately, recent versions of Web browsers do not permit the user to turn off the Web browser’s caching mechanism. Sometimes the caching mechanism can be overridden by forcing the browser to refetch the document from the Web server. To bypass the local cache select **Reload** (Netscape) or **Refresh** (Internet Explorer).

At times, however, refetching the document from the Web server may not work either. The Web browser also caches MIME type information and other *meta-information* about documents. In this case, you must delete all files from the Web browser’s cache to get the correct result. To delete the cache:

• **Netscape:** **Edit → Preferences → Advanced → Cache.**

• **Internet Explorer:** **Tools → Internet Options → Delete Files.**

Before we conclude this section on Web documents, let’s talk about how the Web server software locates a requested file from disk and reads it. The operating system is actually responsible for most of the file management tasks, and the server merely issues calls against the O/S whenever it needs to retrieve a file. The excerpted reading (reprinted at the back of the unit) describes this process.
Reading 2.1

Self-test 2.1

1. Describe the steps the Web server completes in serving a document.

2. How does the Web server make a network connection and find the document to serve over the network?
Document organization

You have been examining how the Web server software uses the MIME-type system to classify documents according to their contents. Now we’ll look at how a Web server stores and manages the collections of individual documents hosted on it. In this section you will see how collections of Web documents are structured and managed as a document tree. The important concepts to understand are:

- tree structure;
- relative name; and
- ‘mirrored’ or replicated document tree.

Document trees

Conceptually, the documents on a Web server are organized into a tree-like hierarchy, similar to the hierarchical file systems in our own computers. The document tree starts out with a root directory which contains files or other subdirectories. In turn, these subdirectories can contain further subdirectories and files.

Every document must have a unique name so that it can be retrieved unambiguously. The unique name is made up of the entire path that must be traveled in order to reach the document, starting from the root. From the figure above, you can see that the complete name for lu_xun.html would be:

/books/authors/lu_xun.html
Two or more documents may have the same file name, as long as their complete names are different. Again, referring to the previous example, you can see that there are three documents named index.html with different absolute path names:

- index.html
- /books/authors/index.html
- /books/titles/index.html

**Referencing documents using hyperlinks**

How do we reference documents using hyperlinks in our HTML pages? Back in *Unit 1*, we described the anchor tag `<A>` and how its `HREF` attribute contains the URL of the desired document. A URL may use the full name of the document, such as:

http://www.abcbooks.com.hk/books/authors/lu_xun.html

or the relative name of the document, depending on where it is referenced from. Here’s the relative link if `lu_xun.html` referenced from within the *authors* folder:

./lu_xun.html

And here’s the relative link referenced from within the *titles* folder:

../authors/lu_xun.html

Relative links are recommended for inline images and other embedded media within an HTML page. In fact, relative links can save you a lot of time if you ever need to transfer your website from one machine to another. You simply retain the existing website directory structure when you copy over the files, and the links will start working right away. It also saves you a lot of typing 😊.

We’ve just described the organization of documents on a Web server as it appears to a Web browser. However, this virtual tree-like hierarchy does not always correspond directly to the physical organization of data on disk. Servers can usually be configured to map parts of the virtual hierarchy to different areas of the physical file system. This is due to the following reasons:

1. **Size** — documents may not fit within a single file system or even on a single computer.

2. **Performance** — documents may be distributed across several computers in order to improve response times.

3. **Ownership** — documents may be owned and maintained by different departments within a company, and accordingly, they may be hosted on different machines.
The following figures show three alternative physical document organizations for a Web server.

**Figure 2.5** One server containing one document tree
*Source:* Yeager and McGrath 1996, 29, fig. 2.2a.

**Figure 2.6** Two servers with one document tree split between them
*Source:* Yeager and McGrath 1996, 29, fig. 2.2b.
 REGARDLESS OF HOW DOCUMENTS ARE PHYSICALLY ORGANIZED ON THE WEB
SERVER’S FILE SYSTEM, THE END RESULT LOOKS THE SAME TO THE WEB USER. EACH
DOCUMENT ORGANIZATION HAS ITS OWN ADVANTAGES. TO DEMONSTRATE THIS
CONCEPT, LET’S EXPERIMENT WITH AN ALTERNATE DOCUMENT TREE ORGANIZATION
USING OUR APACHE WEB SERVER.

In Activity 2.2, you configured two Web servers listening on two
different TCP/IP ports. These two Web servers both had the same
document tree root — the top node of the document tree. In the following
activity we will start two Web servers again. This time the second Web
server will have a different document tree root node.

Activity 2.4

1 You previously configured a Web server to listen to TCP/IP port
8080. We will now edit the configuration file for this server
(httpd8080.conf) so that it uses a different document root from
our original Web server at port 80.

We need to modify the DocumentRoot directive in
httpd8080.conf:

    DocumentRoot $(Apache_rootdir)/htdocs2

where $(Apache_rootdir) is the directory where the Apache Web
server is installed.

2 Save httpd8080.conf in the conf directory. Now create the
directory htdocs2 under $(Apache_rootdir). We need to put a
document in this newly created directory, so just create a simple
webpage called index.html which has a level-1 heading <H1>
containing the text I am index.html from htdocs2. Save this new version of index.html in the htdocs2 directory.

3 Now start the two Web servers — one listening to port 80 and using the default configuration file (conf/httpd.conf), and another one listening to port 8080 and using another configuration file (conf/httpd8080.conf).

   The command for starting a Web server and passing the name of a configuration file to it is

   % httpd -f $(Apache_rootdir)/conf/httpd8080.conf

   where httpd8080.conf is the configuration file instructing the server to listen at port 8080 and $(Apache_rootdir) is the directory where the Apache Web server is installed.

4 Retrieve the default homepage on the original Web server:

   http://localhost/index.html

   This page should display the original homepage installed by Apache (unless you’ve modified it).

5 Retrieve the same Web document on the Web server listening to port 8080 and using a different document tree:

   http://localhost:8080/index.html

   This page should display a heading that says I am index.html from htdocs2.

You should now understand how the Web document tree is organized and how the MIME-document type system works. Do the following self-test to test your knowledge of these concepts.

---

**Self-test 2.2**

1 What are the MIME type and common file extension(s) for an HTML document?

2 Name two file extensions and corresponding MIME types that are natively supported in Web browsers.

3 Name four file extensions, their corresponding MIME types, and external application programs.

4 Describe three ways of organizing and distributing a document collection using two Web servers.

5 What is the advantage of serving a single Web document tree with two Web servers as you configured in Activity 2.3?
This concludes the general overview of each of the components in the equation

Web server = computing platform + Web software + documents.

The documents reside on the computing platform, which provides the file system and network software used by the Web server software to locate and transmit documents over the network. Let’s now take a closer look at how the Web server processes a request after it comes in through the network port.
Web server process models

In the previous section, ‘Web server software’, you learned that the Web server performs the same task repeatedly — accepting and serving all incoming requests for documents on the network port assigned to it. So does this mean that Web server software is trivially simple? Not quite. In describing our step-by-step examples of the Web server software, we left out some crucial hidden details.

The most important detail is that the Web server does not process requests in a sequential order — one at a time. This method is much too slow for a busy server! This approach is also inefficient since requests for smaller documents might be delayed while a request for a large file is being processed. User requests might form an increasingly longer queue as the server finds itself overwhelmed by an ever-growing backlog.

In this section, you will learn how the Web server is designed to handle multiple user requests in parallel, otherwise known as multiprocessing. We will discuss the different process models that allow Web servers to process more than one request at a time, namely:

1. cloning a copy of the httpd program for each request;
2. multithreading the httpd program; and
3. spreading the work among several helper programs.

At the end of the section, you will experiment with your Apache Web server to discover how it records Web server process model events.

The next reading discusses some possible Web server process models and how they work. The virtual host concept and details on establishing TCP/IP connections are also important. You can find the reading at the end of this unit.

Reading 2.2


Apache 2.0, the local Web server used in MT834, allows you to choose the process model that is most suitable for your needs. This is indicated within the configuration script when the server is initially compiled. If you installed your Web server using a compiled version, then you are using the default model for your operating system.

On UNIX, the default is the non-threaded, pre-forking parent-child process model where a single parent process launches multiple child processes which listen for connections and serve them when they arrive.
Root  19481  0.0  0.2 21124 5908 ? S Nov05  0:00 /usr/sbin/httpd
Apache 13947  0.0  0.3 21412 7168 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13948  0.0  0.3 21376 6588 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13949  0.0  0.3 21396 7144 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13950  0.0  0.3 21404 7144 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13951  0.0  0.3 21396 7152 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13952  0.0  0.3 21412 7156 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13953  0.0  0.3 21396 7136 ? S 04:02  0:00 /usr/sbin/httpd
Apache 13954  0.0  0.3 21404 7140 ? S 04:02  0:00 /usr/sbin/httpd
Apache 28682  0.0  0.3 21388 7192 ? S 05:10  0:00 /usr/sbin/httpd
Apache 28683  0.0  0.3 21396 7216 ? S 05:10  0:00 /usr/sbin/httpd
Apache 28684  0.0  0.3 21376 6632 ? S 05:10  0:00 /usr/sbin/httpd
Apache 28985  0.0  0.3 21412 7212 ? S 08:05  0:00 /usr/sbin/httpd
Apache 29011  0.0  0.3 21376 6632 ? S 08:22  0:00 /usr/sbin/httpd
Apache 29012  0.0  0.3 21404 7200 ? S 08:22  0:00 /usr/sbin/httpd
Apache 29013  0.0  0.3 21396 7220 ? S 08:22  0:00 /usr/sbin/httpd

Figure 2.8  Screenshot from a Linux machine showing the parent Web server process (owned by user ‘root’) and multiple child processes (owned by user ‘apache’) launched by the parent

On Windows, the default is the multithreaded process model where a single parent launches a single child process which in turn creates threads to handle requests.

In the next activity, you will try to find out which process model is being used by your localhost Apache Web server.

Activity 2.5

1 The following URLs on your localhost Web server display statistics illustrating the Web server’s process model. However, these URLs may not work straightaway without changes to the configuration file (shown in steps 2 and 3).

   http://localhost/server-status
   http://localhost/server-info

2 mod_status and mod_info can be compiled into Apache or loaded dynamically through the LoadModule directive in httpd.conf in order to view the pages above. To load these modules in, make sure these lines are present in your configuration file:

   LoadModule status_module modules/mod_status.so
   LoadModule info_module modules/mod_info.so

3 Location handlers must be set up for the /server-status and /server-info URLs. You should also create an empty folder called server-status under htdocs.
While viewing the two URLs mentioned above, look for these Web server process model terms defined in Sections 2.4.2–2.4.3 of Reading 2.2:

- child;
- daemon;
- thread;
- keep-alive;
- number of children serving requests;
- number of requests served by each child;
- number of idle children; and
- multithreading.

In your text editor, view the file 
$(Apache_rootdir)/logs/error.log. This log records how the Web server parent process creates a child process and hands it a network connection.

Advantages and disadvantages of different process models

Each copy of the Web server that is currently running uses up system resources, even when it is merely listening for requests. On the other hand, initiating a new server process for every incoming request poses a considerable overhead on the system, since the server configuration files must be read and processed each time.

It is generally recommended to start an initial number of Web servers when the system boots up rather than as they are needed. Requests can be served from within the pool of running server processes, and new ones are spawned only if the current pool is unable to meet demand. If traffic is too light and there are more processes running than is needed, child
processes may also be killed by the parent to avoid wastage of system resources.

Process-only servers still incur considerable overhead as the shared CPU switches from one process to another. Multithreading could be more efficient because the same copy of the Web server is able to process multiple requests. The operating system executes different parts of the same program, called threads, simultaneously. Switching between threads requires less overhead than switching between processes. However, multithreaded servers are more complex to write because they must be designed in such a way that all threads can run at the same time without interfering with each other.

Complete the following self-test to ensure you have understood the important concepts in the Web server process model.

**Self-test 2.3**

1. Name three ways to design a Web server process model to handle multiple simultaneous requests.

2. Describe some advantages and disadvantages of each process model.

3. What is virtual-host support? Describe one situation where virtual-host support could be used.

4. Describe how TCP/IP reads and writes data across a network connection. What does TCP/IP do if there is an error during transmission?

---

**The Web browser: some hidden details**

Web browsers also have some hidden details and extra capabilities. You have examined how a Web browser requests single GIF and JPEG images and HTML pages. But how does the Web server handle multimedia documents where images are mixed with text? These documents are called *inline images*.

**Inline images**

In the next reading, you will see how the Web browser handles inline images. Web browsers can also support display and formatting for Internet protocols other than HTTP, such as File Transfer Protocol (FTP). The important concepts to consider in this reading are inline images and access to FTP. You can find the reading at the end of this unit.
Reading 2.3


You will now look at two simple examples that illustrate our Web browser’s hidden capabilities. First, let’s examine the way our Web browser handles inline images served by the Web server on your local computer.

Activity 2.6

1 You should have a Web server running at port 80 as specified in httpd.conf. The command for starting your Web service if it is not already started is

   % httpd -f $(Apache_rootdir)/conf/httpd.conf

2 Retrieve the ABC Books homepage with your Web browser at http://localhost/abc_home.html.

3 View the HTML page source for abc_home.html with your browser. You will see that it contains multiple inline images. Here are a few examples:

   IMG SRC="abc_logo.gif"
   IMG SRC="10050.gif" (e.g., the book cover for product ID 10050)
   IMG SRC="10060.gif" (e.g., the book cover for product ID 10060)

   … and so on.

   The Web browser knows how to interpret inline image anchors and automatically formulates multiple HTTP requests to send to the Web server. The first HTTP request retrieves abc_home.html. The second HTTP request retrieves abc_logo.gif. The third HTTP request retrieves 10050.gif. Further requests are sent until all the inline images have been retrieved.

4 View the Web server log file at $(Apache_rootdir)/logs/access.log to see how the browser communicated with the Web server multiple times to retrieve all objects needed within one URL. The transactions that were generated by requesting http://localhost/abc_home.html may look similar to this:

   127.0.0.1 - - [07/Nov/2003:14:57:45 +0800] “GET /abc_home.html HTTP/1.1″ 200 1358
127.0.0.1 - - [07/Nov/2003:14:57:45 +0800] "GET /images/10050.jpg HTTP/1.1" 200 7200
127.0.0.1 - - [07/Nov/2003:14:57:45 +0800] "GET /images/10060.gif HTTP/1.1" 200 7802
127.0.0.1 - - [07/Nov/2003:14:57:46 +0800] "GET /images/10070.jpg HTTP/1.1" 200 7484
127.0.0.1 - - [07/Nov/2003:14:57:46 +0800] "GET /images/10080.gif HTTP/1.1" 200 7913

... and so on.

FTP

Now, let’s examine the way our Web browser handles other Internet protocols, like FTP.

Activity 2.7

1. Enter the following URL-style requests into your Web browser to see how your Web browser acts as an FTP client:

   ftp://sunsite.ust.hk

   ftp://ftp.microsoft.com

2. Compare and contrast the result of calling these same FTP servers with a native FTP client on your machine. Type this on the command line:

   ftp ftp.microsoft.com

3. When prompted, enter anonymous as the log-in name and your email address as the password. Your screen should look similar to this:

   Connected to ftp.microsoft.com
   220 Microsoft FTP Service (Version 5.0)
   Name (ftp.microsoft.com:Administrator): anonymous
   331 Anonymous access allowed, send identity (e-mail name) as password.
   Password: student@ouhk.edu.hk
   230-This is FTP.Microsoft.Com.
   230 Anonymous user logged in.
   Remote system type is Windows_NT.
   ftp>

   Once you are in the FTP shell, you can issue file transfer commands by typing them in.

4. When you use your Web browser to call via the FTP Internet protocol, the browser automatically logs you on as ‘anonymous’ to the remote FTP server. The Web browser also enters a dir command
and formats and displays the contents of the FTP server’s top-level directory.

This completes our analysis of how HTTP version 1.1 works, including a few hidden details about the Web server and Web browser. You should have a good overall knowledge of how the Web server and Web browser work together to implement HTTP. In the next section, you will examine the structure and meaning of the HTTP messages exchanged between the browser and server.
HTTP — a closer look

Throughout this course, you’ve been hearing how requests are sent by the browser and how responses are sent back by the server. In this section, we will take a closer look at the rules which govern how these request and response messages are formed. These rules are laid down by the HyperText Transfer Protocol (HTTP), the protocol created specifically for the World Wide Web.

HTTP is an Internet protocol specification produced by members of the Internet Engineering Task Force (IETF). Internet protocol specifications begin as working draft documents or ‘request for comment’ (RFC) documents. The IETF calls them RFCs because it encourages everyone to make comments about working draft documents. When all the comments are reviewed, the Internet draft is approved and becomes a protocol specification.

HTTP version 1.0 was first widely used around 1992. HTTP has now been updated from version 1.0 to version 1.1, and this is the version we will use in MT834. Your local Apache Web server implements a set of basic HTTP/1.1 features so that it can serve HTML documents and images. Netscape and Internet Explorer Web browsers implement a set of basic HTTP/1.1 features so that they can retrieve HTML documents and images. In fact, all Web browsers and Web server software must implement a core set of features from the HTTP protocol. The HTTP/1.1 protocol specification is very large and contains many optional features. In practice, no Web browser or Web server implements the full, complete HTTP protocol specification.

Some older Web browsers and Web server software still implement only version 1.0. When a Web browser and a Web server do not have the same HTTP version and/or HTTP features, communication between the browser and server can be poor and error-prone. Usually a Web server must be designed to handle the capabilities and limitations of the different Web browsers that might interact with it.

From our sneak preview of HTTP back in Unit 1, you know that an HTTP message consists of several lines of text. The message itself is divided into a header and a body. There are cases when the body is optional, but the header is always required. If present, the body is separated from the header by two carriage returns, which show up as a blank line in the message. The following figure illustrates the basic structure of an HTTP message.
An HTTP transaction consists of the request sent by the client and the response returned by the server. Each file needed to construct a webpage must be requested separately, which means that the browser must send multiple HTTP requests in order to render an HTML document containing inline images or embedded multimedia. Depending on the HTTP version used, a series of connections (HTTP/1.0) or a single connection (HTTP/1.1) is needed for transmitting the entire set of requests and responses that comprise a single HTML document.

During a request for a Web document, the Web browser and Web server exchange *header* messages. When the Web browser wants to request a document, it sends a message containing the start line and some request headers to the Web server. The Web server replies with a start line which is different from the request, followed by some response headers, and then the requested document if it is available. In the next reading, it is important to note the components and form of each of these header messages. You can find the reading at the end of this unit.

---

**Reading 2.4**


One thing that might be deduced from Reading 2.4 is that HTTP does not retain any knowledge about the clients that it serves. For example, it does not remember if these clients have ever visited the website before, nor does it know which set of requests originated from the same client. HTTP is considered a *stateless* protocol because it does not maintain any information about the state of the transactions it conducts with its clients. The stateless nature of HTTP will be discussed further in *Unit 5 Databases and state management*. 

---

![Figure 2.9](image) Basic structure of an HTTP message
HTTP requests

The HTTP request is sent from the client to the server. Here’s a summary of the different components within an HTTP request:

1. start line containing the requested resource, the HTTP method and version (required);
2. request header containing the domain name the request is sent to (required);
3. request headers containing metainformation about the requesting browser and its capabilities (optional);
4. entity headers containing metainformation about the contents of the entity body or about the requested resource (optional); and
5. entity body containing data sent to the server (included only when the request method is POST or PUT).

Items 1 to 4 are placed in the HTTP header and item 5 is placed in the HTTP body.

As an example, here is the minimal request that must be sent to www.abcbooks.com.hk in order to request the document /products/books/19015.html:

```
GET /products/books/19015.html HTTP/1.1
Host: www.abcbooks.com.hk
```

The first line contains the method GET, along with the Uniform Resource Identifier (URI) of the information requested and the HTTP version. This asks the Web server to retrieve or get the document from the given Web address, using version 1.1 of HTTP.

The second line contains the Host header, which is required under HTTP 1.1. This header indicates the host name or domain name that the request is sent to. It’s possible for a single Web server to host multiple domains (e.g., virtual hosts), so the client must uniquely identify which domain they are communicating with on that server. There will be a more detailed discussion on how this works in a later section called ‘Virtual hosts’.

HTTP responses

The HTTP request is sent from the server to the client. Here’s a summary of the different components within an HTTP response:

1. start line indicating the status (e.g., success or failure) of the request (required);
response headers containing metainformation about the Web server, its configuration and the methods that it supports. This could also request authorization information or tell the client to try resending the request at a later time (optional);

entity headers containing metainformation about the contents of the entity body, such as encoding schemes, length, type, and origin (optional); and

entity body containing the resource sent back by the server (optional).

Here is the response that is received from www.abcbooks.com.hk when requesting the document /products/books/10050.html:

HTTP/1.1 200 OK
Date: Mon, 06 Dec 1999 20:54:26 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Fri, 04 Oct 1996 14:06:11 GMT
Accept-Ranges: bytes
Content-length: 327
Connection: close
Content-type: text/html
GET /products/books/10050.html HTTP/1.1

… followed by the document.

The first line, also called the status line, contains the HTTP version, status code and a status description (code 200 means the document was successfully retrieved). The remaining response and entity headers contain additional information which can help the browser interpret and render the document sent to it.

Most modern browsers and servers now use HTTP 1.1. In the activity, we will skim through the official HTTP/1.1 specification and identify the header fields that are formally allowed in HTTP messages.

Activity 2.8

Before you begin, please download the HTTP/1.1 specification from http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf.

Now go through the request headers that may be found in an HTTP/1.1 request (RFC2616, Section 5.3, page 26, ‘Request header fields’). Compare and contrast these headers to the ones listed in HTTP/1.0, back in 1992 (http://www.w3.org/Protocols/HTTP/HTTRQ_Headers.html).

HTTP originally did not have any provision for sending the host name that a request was intended for. That’s why the Host header does not appear in the 1992 version. The effect was that every
website needed to have its own IP address. Since there is a limit to
the number of unique IP addresses available (approximately 4 billion
under IP version 4), HTTP/1.1 was modified to allow name-based
virtual hosting via the Host header.

Another thing to remember is that an HTTP request can be a simple
request or a full request. A full request will include all optional
headers, but it is not necessary to send a full request every time.

Next, go through the response headers that may be found in an
HTTP/1.1 response (RFC2616, Section 6.2, page 28, ‘Response
header fields’ and Section 7.1, ‘Entity header fields’).

Compare and contrast these response headers to the ones listed in
HTTP/1.0, back in 1992 (http://www.w3.org/Protocols/
HTTP/Object_Headers.html).

In 1992 the Content-Type header was optional. As HTTP matured,
header fields such as Content-Type became required so that Web
browsers could receive additional metainformation about the data
sent back by the Web server.

Download RFC 2045 from http://www.ietf.org/rfc.html (type RFC
2045 into the search box). This retrieves the Multipurpose Internet
Mail Extension (MIME) specification, ‘Part one: format of Internet
message bodies’.

Skim-read Section 5.0, page 10, on the ‘Content-Type header field’,
Section 5.1, page 12, on ‘Syntax of the Content-Type header field’,
and Section 5.2, page 14, on ‘Content-Type defaults’.

The important concept to learn is that HTTP Content-Type headers
were derived from the Content-Type headers defined in the MIME
standard. Also, the MIME standard states that ASCII text is the
default Content-Type if no Content-Type is sent or found. We
witnessed this to be true in our local Web server implementation.
When we requested abc_home.pdf and the PDF MIME type was
not found, the document was displayed using the default media type
— ASCII text.

The ‘optional’ nature of the request header fields, response header fields
and the entity header fields has caused the wide variation between the
implementations of HTTP 1.1 in Web browsers and Web servers. Also,
the number and type of request header fields and entity header fields
have evolved from HTTP version 1.0 to HTTP version 1.1. In the future
Web browsers and Web servers may be required to support a greater
number and variety of header fields.
Manually issuing HTTP requests using telnet

Now that we are familiar with the sequence and content of HTTP header messages, let’s experiment with our localhost Apache Web server implementation of the HTTP protocol to see how the communication exchange works. We will trace through a simple HTTP request/response sequence by requesting the abc_home.pdf document with our Web browser.

The telnet utility lets you open an interactive socket to an HTTP server. Using the command line, you can manually type in an HTTP request, and view the HTTP response written to your screen. You don’t usually get to see these messages getting exchanged when you use the browser. However, manually issuing the HTTP requests by hand lets you see everything that’s going on behind the scenes. It’s also a great help when learning HTTP, so let’s try this out in the next activity.

Activity 2.9

From the command line, open a connection to your local Web server with the telnet client:

```
telnet localhost 80
```

Then enter your request line by line. Press the Enter key twice after the headers have been entered in order to get the blank line:

```
GET /path/file.html HTTP/1.1
Host: localhost
[blank line here]
```

After you finish your request with the blank line, you’ll see the raw response from the server, including the status line, headers, and message body.

Here are some suggested HTTP requests that you can form and send to the server:

1. request abc_home.html (an HTML page);
2. request abc_home.pdf (a PDF document);
3. request a file that doesn’t exist (e.g., notfound.html);
4. request index.html using the HEAD method instead of GET. The server will behave as if it was executing a GET, but will only send the response headers without the actual document. This makes the screen display less cluttered and easier for you to understand;
5. request an invalid method (e.g., instead of GET, use a meaningless method such as MAKE); and
6 enter a request without the Host: header. What happens?

Note: There is feedback on this activity at the back of the unit.

The HTTP protocol sequence and the components of HTTP messages described in Activity 2.9 are the foundation of every HTTP document request. Now complete the following self-test to ensure you have understood these concepts.

**Self-test 2.4**

1 Write a simple HTTP request sent by your browser to request news.html from a Web server. Write the HTTP response sent by the Web server for a successful delivery.

2 Outline the basic structure of HTTP requests and responses using a bulleted list format.

3 Which HTTP method is used to request search results from a search engine such as Google’s dynamic documents?

4 What do you think is the most common HTTP response status code? Why?

5 Name a request method or a header field in the HTTP 1.1 protocol specification that is not supported by most Web browsers or Web servers.
Other HTTP features

In this section, we will explore a few successful feature enhancements that have been added to HTTP since 1992. A successful feature enhancement is one that is implemented by most current Web browsers and servers. The features you will examine are:

1. virtual hosts;
2. content negotiation; and
3. persistent connections.

Virtual hosts

Previously, we learned that virtual-host support permits a single Web server to manage multiple server names which correspond to multiple IP addresses. This is called IP-based virtual hosting and allows the same Web server on the same computing platform to serve different host names. A sample configuration is given in the following table.

Table 2.2  IP-based virtual hosting configuration

<table>
<thead>
<tr>
<th>Host name</th>
<th>IP address</th>
<th>Document root</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.companyA.com">www.companyA.com</a></td>
<td>111.22.33.44</td>
<td>/usr/local/apache/htdocs/companyA</td>
</tr>
<tr>
<td><a href="http://www.companyB.com">www.companyB.com</a></td>
<td>111.22.33.45</td>
<td>/usr/local/apache/htdocs/companyB</td>
</tr>
</tbody>
</table>

In the example above, the machine hosting the two domains (companyA and companyB) also needs to have two network interfaces, one for each IP address (111.22.33.44 and 111.22.33.45). The disadvantage of an IP-based virtual host is that it requires a new network interface for each new network name. Maintaining two or more network interfaces on a single computing platform can be a difficult task and is not permitted by some operating systems. IP addresses are also limited in number, and there’s a danger that we may run out of unique IP addresses if every domain required its own address.

Another type of virtual-host support has been developed to overcome these limitations. In a name-based virtual host there are multiple server names corresponding to a single IP address served by one Web server. This means that one server at one IP address can be multihomed, i.e., the home of several Web domains. A sample configuration is shown in Table 2.3.
The advantage of a name-based virtual host is that it works with only one network interface on the computing platform. But, with the Web server serving more than one name and therefore more than one document root, requests for documents from the Web browser become ambiguous. Several domains living on the same server is like several people sharing one phone. The caller must tell whoever picks up the phone who they’re looking for.

This means the client must specify which host name the request is intended for, using the Host: request header. As discussed previously, the Host field contains the host name where the requested resource is found. For example, a simple request for abc_home.pdf from www.companyB.com would take the form:

```
GET /abc_home.pdf HTTP/1.1
Host: www.companyB.com
```

The server uses the host name sent by the client to figure out the correct document root where it will locate the requested information. The Host: header is required in HTTP/1.1.

## Content negotiation

It’s possible for the same information to be available in different languages and/or media types. It may also be available in compressed or uncompressed versions, or it may combine several of the above features. Content negotiation is the process of a Web server selecting and delivering the version of a document that best matches the Web browser’s display preferences and capabilities.

HTTP allows multiple versions of the same content to be accessed via the same URL. The files where these different versions are stored would have similar names, except for the file extensions. For example, the same image may exist in GIF (e.g., logo.gif) and JPEG (e.g., logo.jpg). The same document may be written in English (e.g., index.html.en) and Spanish (e.g., index.html.es).

When a user accesses a particular URL, the browser sends request headers containing information about which representations they prefer. For example, a browser could indicate that it would like to see information in simplified Chinese, if possible, or else English will do. It could also send the list of media types that it is capable of accepting. A numeric preference factor can also be supplied by the browser to indicate

### Table 2.3  Name-based virtual hosting

<table>
<thead>
<tr>
<th>Host name</th>
<th>IP address</th>
<th>Document root</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.companyA.com">www.companyA.com</a></td>
<td>111.22.33.44</td>
<td>/usr/local/apache/htdocs/documentA</td>
</tr>
<tr>
<td><a href="http://www.companyB.com">www.companyB.com</a></td>
<td>111.22.33.44</td>
<td>/usr/local/apache/htdocs/documentB</td>
</tr>
</tbody>
</table>
which version it prefers over the multiple versions that it is willing to accept.

With content negotiation, the Web server doesn’t have to rely on guesswork when deciding which version of the information to send.

Content negotiation is achieved by exchanging request header fields and entity header fields within HTTP messages. The next reading describes how these headers are formed, along with the different ways in which negotiation takes place — server-driven, agent-driven, and transparent negotiation. It will be immediately followed by a hands-on activity which lets you experience content negotiation in action!

**Reading 2.5**


These request headers are used to describe the capabilities and user preferences of the Web browser:

1. `Accept`
2. `Accept-Charset`
3. `Accept-Language`
4. `Accept-Encoding`

These entity headers are used to describe the media type, language, encoding and length of the data sent by the Web server:

1. `Content-Type`
2. `Content-Language`
3. `Content-Encoding`
4. `Content-Length`

Let’s now try out content negotiation for ourselves! Our local Apache Web server maintains several representations of `index.html`, the default homepage. Under the `htdocs` directory, you will see different language versions of the homepage in files such as `index.html.ca`, `index.html.es`, `index.html.fr` and so on.

In the next activity, we will configure the Web server so that it serves a specific language version by default. We will then override this default by explicitly including the language version we want in our HTTP request headers.
Activity 2.10

1 In your text editor, open the file `index.html.var` from the `htdocs` directory. This is the type map file which contains all the variations that are available for `index.html`, along with information about each variant. Look briefly at the contents of this file and close it without changing any of its contents.

2 In your text editor, open the configuration file `httpd.conf`. Look for the `LanguagePriority` directive, which lists the languages to be used when returning documents, in decreasing order of priority. The language values are indicated by a two character code (e.g., `en` for English, `fr` for French, `de` for German, `it` for Italian, `tw` for Chinese [Taiwan], and so on). By default, English (`en`) is at the front of the list, so let’s move `tw` to the front instead.

   This is how the `LanguagePriority` directive looks like after I’ve edited it in my `httpd.conf`:
   
   `LanguagePriority tw en da nl et fr de el it ja ko no pl pt pt-br ltz ca es sv`

3 Now use the telnet client to set up an interactive session with your local Web server. Type this in the command line:

   `telnet localhost 80`

4 Enter the HTTP request line by line. Press the `Enter` key twice after the headers have been entered in order to get the blank line.

   ```
   GET / HTTP/1.1
   Host: localhost
   [blank line]
   ```

   If all works well, you will get the Chinese (Taiwan) version of the homepage. The name of the file returned will be `index.html.zh-tw.big5`, as shown in the `Content-Location` header returned by the server.

   Why do you think the server chose the Chinese language version to send you?

5 Next, let’s send an HTTP request which explicitly sends the preferred language to the server. We’ll ask for the French (`fr`) version of the homepage this time.

   ```
   GET / HTTP/1.1
   Host: localhost
   Accept-language: fr
   [blank line]
   ```

   What happens?

6 Now, what happens if we send more than one language in the `Accept-Language` header? Let’s request French and English this time.
GET / HTTP/1.1
Host: localhost
Accept-language: fr, en

What happens?

7 The Accept-language header allows you to add a preference factor ranging from 0.00 to 1.00 which tells the server which language is preferred if more than one version is available. Let’s try this out now.

GET / HTTP/1.1
Host: localhost
Accept-language: fr; qs=0.5, tw; qs=0.25, es; qs=1.0

What happens?

8 Now let’s send a language code which doesn’t exist, such as bb.

GET / HTTP/1.1
Host: localhost
Accept-language: bb

What happens? You’ve just experienced content negotiation first-hand. What a fun way to learn a new language!

Note: There is feedback on this activity at the back of the unit.

Persistent connections

HTTP/1.0 opens and closes a new TCP connection for each request and response message exchanged. In reality, most webpages are made up of several files on the same server. An example would be an HTML document with inline images and embedded media. Opening and closing multiple TCP connections associated with serving a single document can lead to network traffic congestion and considerable overhead on CPU time, bandwidth, and memory.

To address these shortcomings, HTTP/1.1 uses persistent connections. This approach leaves the TCP connection open between consecutive operations instead of opening and closing new connections every time. Now all the files needed to render the same Web document can be requested and transmitted within the same connection, leading to faster and more efficient performance.

The next reading taken from the HTTP/1.1 specification describes the purpose of persistent connections in more detail.
Reading 2.6


Note: Read only section 8.1.1 ‘Purpose’ and 8.1.2 ‘Overall operations’.

The next figure illustrates how multiple HTTP transactions (each request-response) can now take place over a single persistent connection instead of over a series of connections.

![Figure 2.10](http://example.com/image.png)

HTTP transactions taking place over a persistent connection

With persistent connections, the browser opens a connection and sends several requests at a time (called pipelining). In turn, the Web server sends responses back in the same order as the requests. Persistent connections are the default under HTTP 1.1. All connections are kept alive unless the client states otherwise, with the following header:

```
Connection: close
```

If this header is sent along with the request, then the server will close the connection following the response, and the client will not be able to use this connection for further requests. The Connection: close header should be used if the Web client does not support persistent connections, or if it knows that this is the last request to be sent out on this connection. The server may also close an idle connection after a suitable timeout period.
Self-test 2.5

1. What are the required header fields sent by the Web browser and the Web server during content negotiation?

2. What are the advantages of using persistent connections?

3. What are the advantages of using virtual hosts? Name two system configurations where one might use this feature. What is the required header field to support name-based virtual-host support?

Aside from the HTTP features we’ve examined here, there are other characteristics that are standard with the HTTP/1.1 implementation, such as chunked encoding, which allows the server to send a document in several parts rather than as a whole. Another feature allows the client to request only a portion of a document. There are also features that will make it easier to implement proxies and caches, which can help reduce network traffic. The HTTP features that are specific to proxies and caches will be discussed further in *Unit 6 Website performance, tuning and caching*. 
Summary

This unit covers the inner workings of the Web server and examines how it uses HTTP to communicate with a Web browser. No matter what your level of involvement with the Web, the concepts in Unit 2 are important to you!

• As a user of Web services, you should understand how MIME extends your Web browser’s capabilities by allowing it to invoke external viewer programs.

• As an application developer, you should understand how MIME extends the Web server’s capabilities by allowing it to serve different content types and take into account user preferences and browser capabilities.

• As a Web service system administrator, you must understand HTTP request and response headers, because you need to know what is taking place on your Web server — which files are the most popular, what Web browsers they are using, what languages they prefer and so on.

• As a Web service provider, you have to organize your tree of Web documents and potentially spread the workload among multiple Web servers for better performance, higher capacity and more reliable Web services.

Unit 2 concludes the coverage of the basic concepts that you will repeatedly draw upon as you progress through the remaining units. The next unit in this course discusses one of the most important Web server functions — the capability to execute scripts that will generate documents in real-time.
Feedback to activities

Activity 2.9
Here’s a minimal set of request and response headers for each request:

<table>
<thead>
<tr>
<th>HTTP request headers</th>
<th>HTTP response headers</th>
</tr>
</thead>
</table>
| GET /abc_home.html HTTP/1.1 Host: localhost | HTTP/1.1 200 OK  
Date: Thu, 06 Nov 2003 13:01:30 GMT  
Content-Length: 1078  
Content-Type: text/html  
[body contains abc_home.html] |
| GET /abc_home.pdf HTTP/1.1 Host: localhost | HTTP/1.1 200 OK  
Date: Thu, 06 Nov 2003 13:01:30 GMT  
Content-Length: 2500  
Content-Type: application/pdf  
[body contains abc_home.pdf] |
| GET /notfound.html HTTP/1.1 Host: localhost | HTTP/1.1 404 Not Found  
Date: Thu, 06 Nov 2003 13:01:30 GMT  
Content-Length: 287  
Content-Type: text/html  
[body contains default page saying file is not found] |
| HEAD /index.html HTTP/1.1 Host: localhost | HTTP/1.1 200 OK  
Date: Thu, 06 Nov 2003 13:01:30 GMT  
Content-Length: 1078  
Content-Type: text/html  
[no entity body] |
| MAKE /index.html HTTP/1.1 Host: localhost | HTTP/1.1 501 Method Not Implemented  
Date: Sat, 08 Nov 2003 07:43:43 GMT  
Allow: GET,HEAD,POST,OPTIONS,TRACE  
Content-Length: 303  
Connection: close  
Content-Type: text/html  
[body contains default page saying HTTP method is invalid] |
| GET / HTTP/1.1 | HTTP/1.1 400 Bad Request  
Date: Sat, 08 Nov 2003 07:45:02 GMT  
Content-Length: 297  
Connection: close  
Content-Type: text/html  
[body contains default page saying the server could not understand the request] |
### Activity 2.10

<table>
<thead>
<tr>
<th>Accept-header value</th>
<th>File returned/Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>index.html.zh-tw.big5</td>
</tr>
<tr>
<td></td>
<td>Since no preference was specified, the language with the highest priority (tw) in the server configuration file is used (LanguagePriority directive).</td>
</tr>
<tr>
<td>fr, en</td>
<td>index.html.en</td>
</tr>
<tr>
<td></td>
<td>en has a higher priority than fr in the server configuration file (LanguagePriority directive).</td>
</tr>
<tr>
<td>fr; qs=0.5, tw; qs=0.25, es; qs=1.0</td>
<td>index.html.qs</td>
</tr>
<tr>
<td></td>
<td>qs has a higher preference factor</td>
</tr>
<tr>
<td>bb</td>
<td>index.html.zh-tw.big5</td>
</tr>
<tr>
<td></td>
<td>There is no document available in this language (bb), so the language with the highest priority is used (tw).</td>
</tr>
</tbody>
</table>
Suggested answers to self-tests

**Self-test 2.1**

1. A Web server listens for requests on the network. When the Web server receives a request for a document from a Web browser, it must locate that document in the file system of the computing platform. When the correct document is located, the Web server sends that document back over the network connection to the Web browser.

2. The Web server listens for requests on a TCP/IP port. When a Web browser wants to request a document, it sends the request to the Web server’s assigned port. The port is an abstraction that provides a simple way to make and use a network connection. In the network address 127.0.0.1:8099, 8099 is the TCP/IP port.

The Web server uses the computing platform’s native file system to locate and return the requested document. A file system is a component of the computing platform’s operating system.

**Self-test 2.2**

1. `.html` or `.htm` is the file extension for an HTML document. The MIME type for an HTML document is `text/html`. On most computing platforms, the file extension for an HTML document is `.html`. The file extension for HTML documents on DOS and Windows systems is `.htm` because DOS only supports file extensions of three characters or less.

2. GIF and JPEG images are natively supported by Web browsers. GIF images have a file extension of `.gif` and a MIME type of `image/gif`. JPEG images have a file extension of `.jpg` and a MIME type of `image/jpeg`.

3. There are many possible answers. A few answers are included in the following table.

<table>
<thead>
<tr>
<th>External viewer</th>
<th>MIME type</th>
<th>File extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macromedia Shockwave</td>
<td>application/x-shockwave-flash</td>
<td>.swf</td>
</tr>
<tr>
<td>Flash player</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Live Audio player</td>
<td>audio/wav</td>
<td>.wav</td>
</tr>
<tr>
<td>Microsoft Excel</td>
<td>application/vnd.ms-excel</td>
<td>.xls</td>
</tr>
<tr>
<td>Quicktime</td>
<td>video/quicktime</td>
<td>.mov, .qt</td>
</tr>
</tbody>
</table>

4. Here are three configurations using two Web servers and one document tree:

- Two Web servers on one computing platform with the whole document tree replicated. The two copies of the document tree
are identical. Each Web server has its own full copy of the entire document tree. The Web servers have the same IP address; but each Web server listens to its own TCP/IP port number.

- Two Web servers on one computing platform with the document tree split between the two Web servers. There is only one copy of the document tree. Each Web server has a different document tree root node. The Web servers have the same IP address, but each Web server has its own TCP/IP port number.

- Two Web servers, each on their own computing platform. The document tree is replicated, with one full copy of the document tree on both computing platforms. Each Web server has the same document tree root node. The Web servers each have different IP addresses and port numbers.

5 Some Web documents are more popular than others. The advantage of serving a single Web document tree with two Web servers is that a popular document can be made more easily available on the Internet. Popular documents can be served faster because there are two Web servers serving the document. Web browser clients have two network ports by which they can reach the document. The workload in serving the document to the Internet is distributed between two Web servers.

**Self-test 2.3**

1 Three Web server process models are:

- parent/child cloning: for each request the httpd ‘parent’ clones itself creating a new ‘child’ process that handles the request;

- multiple threads of execution, ‘multithreaded’. Only one copy of the process is needed to serve multiple requests, with different parts of the same program (e.g., threads) running at the same time; and

- multiple ‘helper’ processes: a dispatcher process spreads the workload among a set of helper processes. The dispatcher process acts as the parent, starting new processes or killing off running processes according to real-time Web traffic volumes.
2 The following table lists some advantages and disadvantages for each process model.

<table>
<thead>
<tr>
<th></th>
<th>Parent/child cloning</th>
<th>Multithreading</th>
<th>Multiple ‘helper’ processes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td>Simple to write and implement.</td>
<td>Switching between threads is more efficient than switching between processes.</td>
<td>Serving requests from a pool of running server processes incurs less overhead than starting a new process for every request.</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>It’s very inefficient to start and stop a new server process for each incoming request. Most modern Web servers no longer use this model.</td>
<td>Multithreaded Web servers are more complex and difficult to write.</td>
<td>There is still overhead involved in switching the CPU from one process to another. The pool of running server processes will still use up system resources even if they are idle.</td>
</tr>
</tbody>
</table>

3 Virtual-host support permits a single Web server to manage multiple server names and multiple IP addresses. This is often used by ISPs (Internet Service Providers) when they offer shared Web hosting accounts on the same machine.

4 TCP/IP breaks the data stream of bytes into discrete packets of data. Each data packet is in an ‘envelope’; it has a header message which describes where the data packet is to be sent across the network. If a data packet is lost or an error occurs, TCP/IP must resend or retransmit the packet of data across the network. On the other end of the network, TCP/IP must receive, decode, and reassemble the individual data packets. Finally, on the client side the sequence of data received is the same as the sequence of data sent over the network.

**Self-test 2.4**

1 Here’s a simple HTTP request to retrieve news.html:

```
GET /news.html HTTP/1.1
Host: www.abcbooks.com.hk
```

And the Web server response may look like this:

```
HTTP/1.1 200 OK
Date: Thu, 06 Nov 2003 13:01:30 GMT
Content-Location: news.html
Last-Modified: Tue, 13 May 2003 07:17:46 GMT
Content-Length: 1078
Content-Type: text/html
Content-Language: en
```

Where 200 means OK, the document was returned successfully. The file size is 1078 bytes.
2 a Basic structure of an HTTP request:
   • start line containing METHOD, Resource and HTTP Version;
   • host header;
   • other client generated headers; and
   • request body.

b Basic structure of an HTTP response:
   • status line;
   • server generated headers; and
   • data.

3 POST is the method used to request search results from a search engine. The POST method allows data to be submitted to a server along with a request, and this method is used to request dynamic documents generated in real-time via server-side scripts. The POST method will be discussed in more detail in Unit 3 Server-side programming.

4 200, i.e., OK, is the most common status code. We are assuming that most documents are successfully retrieved, or else the World Wide Web would not be as successful as it is today 😊.

5 PUT and DELETE are two request methods that are not supported by most Web servers. Web servers typically do not allow PUT and DELETE because these allow users to update and delete a particular file within the document tree. These operations should be restricted to authorized users.

Self-test 2.5

1 The header fields that may be sent by the browser during content negotiation are: Accept, Accept-Language, Accept-Charset, and Accept-Encoding. The header fields that may be sent by the Web server during content negotiation are: Content-Type, Content-Language, Content-Encoding, and Content-Length.

2 Persistent connections are efficiently used by the Web browser for retrieving a series of files from the same Web server. Persistent connections save the time needed to create and dismantle a TCP/IP connection for each request. Sending files over a persistent connection is faster since the time needed to create each TCP/IP connection is saved. When a TCP/IP connection is created, memory is allocated and a series of handshake messages between the client and server are exchanged, which takes time. CPU time and memory on the host is saved. A Web browser that supports persistent connections may pipeline its requests to increase efficiency and save time. Pipelining involves sending multiple requests without waiting for the Web server to respond.
The advantages of using virtual hosts are that:

- the same machine can be used to serve different websites even though the host has only one network interface; and
- Only one Web server daemon is needed to serve multiple websites. This reduces the workload of the server as compared with running two Web server daemons.

Virtual-host support can be used as:

- a Web server with IP-based virtual-host support on a computing platform with multiple network interfaces (and therefore multiple IP addresses). The Web server maintains multiple host names, each one corresponding to a different IP address; and
- a Web server with name-based virtual-host support on a computing platform with one network interface (and therefore only one IP address). The Web server maintains multiple host names which map to the same IP address.

The Host header field is required to support name-based virtual-host support. This header should always be present for HTTP/1.1 implementations.
References


