Universidad Politécnica de Cartagena

School of Telecommunications Engineering

APPLICATIONS ON THE INTERNET

Lab 1: HTTP Server. Apache 2

Teachers:

Esteban Egea Lopez

1. GOALS

- Understand the operation of an HTTP server.
- Understand how documents are organized on a Web server.

2. INTRODUCTION

An *HTTP server or web server* is an application that handles content using the HTTP (Hyper Text Mark-up Language) protocol. The server accepts client requests and returns the appropriate HTML documents. There are also a number of complementary technologies that are responsible for increasing the server capabilities beyond its ability to deliver standard HTML pages, such as CGI (Common Gate Interface), SSI (Server Side Includes) or PHP are (Hypertext Preprocessor), which enables it to provide dynamic content HTML documents.

Apache (http://www.apache.org) has been the most popular web server since 1996, currently used by more than 50% of websites on the Internet. It is a freely distributed software application that implements an HTTP server, available for several operating systems. The source code is available.

Simplistically, Apache is a process running in a UNIX environment. It is a "demon" that runs in the background and is constantly listening port 80, in its default configuration, waiting to serve HTTP requests.

2.1 Modules and multiprocessing

Apache is designed to be a powerful and flexible web server that can run on the widest variety of platforms and environments. Differences between platforms and environments make different features or functionality often necessary, or the same feature or functionality is implemented differently for greater efficiency. Apache uses a **modular** design. This design allows website administrators to choose which features will be included in the server, selecting modules to load either at compile or runtime

Apache 2.0 extends this modular design to the most basic functions of a web server. The server comes with a series of **MPM (multiprocessing modules)** which systemctl are responsible for listening the network ports on the machine, accepting requests, and dispatching children processes to handle serving.

The extension of the modular design to this level server provides two important benefits:

- Apache can handle more easily and efficiently a wide variety of operating systems. Specifically, the Windows version of Apache is much more efficient, since mpm_winnt can use native networking features instead of using the POSIX layer as does Apache 1.3. This benefit also extends to other operating systems that implement their respective MPMs.

- The server can be better customized for the needs of each site. For example, websites that need more than anything scalability can use a MPM basen oon *threads* like *worker,* while sites requiring mainly stability or compatibility with older software can use *prefork*.

At the user level, MPMs are like any other Apache module. The most important difference is that only one MPM can be loaded on the server at any given time.

There are several available MPM modules, some are specific to the platform, we focus on two of the available for UNIX:

- *Worker:* the server is implemented as a hybrid multi-process multi-threaded application.. By using *threads* to serve requests, it is able to serve a large number of requests with fewer system resources than a process based server. However, it retains much of the stability of a process based server maintaining multiple processes available, each with many threads.
- *Event:* a variant of the previous allowing delegate certain tasks processed common supporting threads, freeing the main threads to handle new requests. Thus scalability is increased.
- *Perfork:* it is the classic server based on multiple processes. It is less efficient but more stable than the aforementioned. Does not use threads, so it can be used with libraries which do not support threads. Furthermore, by isolating requests prevents a thread problem affects others processes..

In this lab we will use the MPM *event* module, which is installed by default. The operation is as follows: when Apache is executed, a parent process, called control, is created, handling requests to port 80. This process, in its turn, launches several child processes *(fork())* that are responsible for listening on the same port and serve requests. Each child creates a fixed number of threads responsible for handling requests, plus a thread that is responsible for listening and managing connections. This is the operation of the module *worker.* The particular operation of *event* is a variation that solves specific problems for Apache. You can check its operation in detail in the Apache documentation.

The operation of the modules and the rules they have to apply to request are determined by **directives** that are included in the **configuration** files. These policies determine, among other things, the directories where documents are going to be served, the access control to apply to requests, etc.

It is usefut to know the user under which the processes and privileges are run. The parent process runs as *root* since it must bind*(bind)* to port 80, which is a privileged port in UNIX. However, **the children processes run under a less privileged user** (usually the user *www-data)* to prevent a process with *root* privileges handle requests that arrive through the network.

The configuration directives set privileges for the children of Apache. Thus, when the parent process creates children, it assigns the user, *www-data,* which has virtually no privileges on system resources. The children processes must have permission to read the contents to be served but none else besides these.

The privileges set by the above directives are also those who are inherited by the *scripts* and system calls and resources used by script code, ,ie they will run with very few privileges.

2.2 Processing requests

The server serves HTTP requests that arrive to port 80 by default. Requests are made using the HTTP protocol. That is, the client sends the server a message encapsulated in a TCP / IP the OS decapsulates the packet and passes it to the server process, a message with the following aspect:

```
GET /index.html HTTP / 1.1
```

This message tells the server (actually, one of the "children" processes) that it looks for the document *index.html,* to be found in the **root directory of documents.** The root directory is the place to find HTML documents that can be served by the site. The directory tree with the available documents to be served is in this folder.

The request string that appears after the GET method, in this example /index.html, is called the **local path**.

The server concatenates the path of the root directory with the local path to obtain the **absolute path of the document**.

Absolute path = root path + local path

The root directory can be any directory on the machine on which Apache runs. In the configuration file the directive *DocumentRoot* indicates this directory.
For example, if the value of this directive is */usr/doc/htdocs*, the server creates an absolute path equal to */usr/doc/htdocs/index.html* in this example.

If this document is in this directory, **and the server process has permission to read** it, it returns it to the client, which will end the request. Otherwise it will return an error message.

Therefore, a request like *www.midominio.com/index.html* means that you are asking the site for the document *index.html* placed under the root directory.
While a request like *www.midominio.com/usuario/index.html* asks for the document *index.html* that is in the directory *user* which in turn is under the root.

All information on the Apache server can be found at: http://httpd.apache.org/docs/

3. LAB ACTIVITIES

3.1 Server Configuration

In practice, most Linux distributions include Apache along with the rest of the operating system. The difference between distributions is the path where the files are installed. For example, in Ubuntu / Debian, configuration files are stored in the */etc/apache2* directory.

In the lab we use the Apache installation that provides Kubuntu.

Once the server is installed, you must configure its operation. Apache is configured using multiple text files. **These files include commands, called directives, that specify the mode of operation of the server.** Each directive specifies the operation of a feature of the server. Each directive has also a number of options. The server, when started, reads the configuration, sets its operating mode and waits for requests. The configuration files are only read at server startup, so ***if a directive is modified, you have to stop the server and restart again for the changes to take place*.**

The configuration files are in the directory */etc/apache2/.* These files contain the default server configuration.

3.2 Execution of server

The server comes with a default configuration that allows us to test it directly.
To run the server, one normally uses a *script.* Kubuntu can run it as a standard service with *systemctl {start | stop | restart ...} apache2*, a service with a parameter to start, stop, restart or others.

- Stop the server. Run the following command:

```
systemctl  stop  apache2
```

- To run the server:

```
        systemctl start apache2
```

Now the server is again running.

- Check that everything works properly. To do this, open a web browser and request the URL http://localhost . **Why this URL? Use an equivalent URL**

- Check the processes that are running. To do this run *ps aux | grep apache2*. Examine which processes are running and the user who executes them and explain what you see.

- To restart the server (when running) use *systemctl apache2 restart*.

3.3 Configuration files and global settings.

The server is configured via simple text files. These files can be located in different places, depending on how you installed the server. The configuration is normally divided into several smaller files for easier management. These files are loaded via the *Include* directive.

The server is configured by setting **configuration directives** in these configuration files. A **directive is a keyword followed by one or more arguments that establish their value.** Configuration files usually include a comprehensive description of each directive, which will be very useful.

In addition to directives, there are **containers.** These differ from directives in that appear between the characters. <>. A container is used to define an application context of the directives. For example, the container *<Directory "/ var / www / user">* indicates that all directives that appear between the start and end *(</Directory>)* apply only to that directory.

In http://httpd.apache.org/docs/2.4/mod/quickreference.html there is a list of Apache directives. There is a description of them as well as the options will be applied and the context in which they are used.

The syntax is very important, so use the above link to verify what you have written if have doubts. Blanks and line breaks and carriage returns are not taken into account, so you can use them to improve readability when editing the file.

In Ubuntu / Debian distributions the settings are stored in */etc/*apache2.

- Examine the Apache configuration directory.

The main configuration file is called *apache2.conf* and contains global settings. In addition it is responsible for loading other configuration files with *Include* directives.

As noted earlier, Apache is modular and therefore the configuration has been divided into modules, ie., **each module provides its own configuration** file.

Moreover, a server can serve **multiple sites** simultaneously, it is what are called ***virtual*** hosts. This means that a single server can serve different domain names, such as www.ssd.upct.es and www.ai.upct.es.

So the approach used for configuration is to have two types of directories:

1. The directory *mods-available* and *sites-available*, contain the configuration files and virtual modules available for the sites. They are modules that have been compiled and are thus available for use, but are not in use.
2. The directory *mods-enabled* and *sites-enabled*, include the configuration files of the **modules actually loaded and running on the** server.

The idea is **to load/activate a module** simply by **copying the configuration file from mods-available to mods-enabled.** And to disable one must delete the mods-enabled entry.
NOTE: Instead of copying and deleting directly, the distribution provides commands to enable or disable modules: *a2enmod and a2dismod* to disable and enable an Apache module. **Use this method**.

Actually, each module has an associated loading file: *modulo.load* and a configuration file: *modulo.conf.* The first simply forces Apache to load the module and the second establishes the settings. Copy both to activate a module.

- Examine the four directories *available* and *enabled,* opening some of the files and reading what is in them.

3.4 Main Server
The concept of main server arises because Apache lets us define virtual directories, ie, allows the same server to handle requests to different IP addresses (on the same machine, for example if you have multiple network interfaces) or allows that the same site (defined by a single IP address) can be assigned different domain names, served by the same server. That is, in the DNS the same IP has been set for different domains.
Virtual servers are defined by the container. *<VirtualHost>.* In this lab virtual servers will not be studied.

Here there are some of the available directives. First some global configuration options: are set in *ports.conf*.

Listen. It indicates the port to be served. Use the default port, 80.

Another important directive is in the *000-default.conf* file inside *mods-enabled*: *DocumentRoot.* It indicates the root directory from which documents will be served. The default is */var /www/htm*. Any request for a document that is not under this directory will not be served. For example, a request to *www.midominio.com/opt/mm.html* looks for the document *mm.html* under the directory *opt* of the root specified in the directive, that is */var /www /html /opt/*.

- **Which folder is the document root?**

- Create a few HTML documents and copy them to different directories under the document root. Verify that you can get them with the browser.

- Assign 740 permissions to the HTML documents and try to get them. Explain what happens.

While the remaining options are set in *sites-enabled /000-default*.

- Configure the server to listen on port 5050. Note that you have to modify two files to change the port. Restart the server and check this operation using your browser. **What URL have you used?**
- Reassign port 80.

Most directives appear inside containers, as we said.

<Directory dir> defines the directory to which the directives apply. It applies directives and access controls based on directories, for example, who and how you can access a given directory. We will talk about access control later.

The first container shown in *apache.conf* is *<Directory/>*. It establishes the directive that apply to all subdirectories from the *root system* (do not confuse with the root of the documents to be served by the Apache server). This is used to apply very restrictive access conditions which then will be overwritten case to case. This way a global controlled behavior for any directory access is implemented .

The next containe is *<Directory "/var/*www">.The directives here override the ones in the container. *<Directory />*. Thus, while the former only allows for any directory the option *FollowSimLinks,* this new container adds more options like *Indexes* and. *MultiViews.* Go to the Apache documentation to know what those options are.

- Copy any web page to your root directory.
- Remove the index.html from your root folder.
- Remove the option *Indexes from*   */var/www/html* and restart the server. Request with your browser the following documents http://localhost. **What happens?**
- Add the option *Indexes* again and request the root, **what happens in this case?**

Let us continue with the directives. Here there are some of the directive that set up how requests are served. In *dir.conf* you can find:

*DirectoryIndex.*This policy appears within a <*IfModule*mod_dir.c>. That is, applies if the module is loaded. Indicates the name of the HTML page that appears as default index, in order of preference from right to left.

```
DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm
```

- HMTL Create 2 pages and save them as index.html and index.htm and request http: // localhost

Within the module *alias.conf one* can find:

Alias. This directive also appears inside the container *<IfModule alias_module>.* Used to indicate that certain directories contain documents to be served even though they are not under the document root (specified in *DocumentRoot)*

For example,directive *Alias /icons/ "/usr /local/web/apache/icons"* is saying that when there are requests for the */icons* directory, they are looked up in */usr/local/web/apache/*icons.
So if there is not an alias, the request *www.midominio.com/icons/foto.gif* makes the server look for a file called *photo.gif* in the */var/www/*icons, ie under the root. Whereas if there is an alias, the file is searched in */ usr / local / web / apache / icons.*

After this directive, you can find a corresponding container for the alias directory establishing access control and configuration of that directory.
Aliases are particularly useful in combination with the user documents, as discussed later.

3.5 User Documents

In the previous section you just checked that the documents that can be served by Apache must be in the document root directory specified in the *DocumentRoot.* This way of organizing documents is suitable for a web site in which there is only one administrator who is responsible for publishing the contents.
A practical problem occurs when there are many users who need to publish documents. Think of the University, for example, if every teacher wants to publish the content of their subject, they would have to send them to the webmaster who has to save them and organize them into directories under the root. And there are more than 500 teachers in the UPCT !. It would also have update then every now and then, etc.

To solve this problem the user should be able to publish and organize his own documents without the administrator. The solution is simple: each user is assigned a directory where he can place documents to be served, but not under the DocumentRoot.

Apache allows to set these directories using the directive *UserDir* whose value is *public_html,* located in the *userdir.conf* module.
When you issue requests of type [*www.midominio.com/~user/index.html*](www.midominio.com/~user/index.html), the server looks in the user directory */home/user/public_html* for the requested file That is, when in the request URL, after the domain symbol, it appear a ~ followed by a **username that exists in the system**, the server searches the document requested under the directory *public_html* of the corresponding user.

Examples:
www.ait.upct.es/~eegea/productos.html return the file *products.html* located in */ home / eegea /public_html*.
www.ait.upct.es/~jvales/imagenes/foto.jpg return the file *foto.jpg* located in the */ home / jvales / public_html / images* directory.

- Create a user (with *adduser* command).
- Check that the *userdir* module is enabled.
- Place HTML documents in the *public_html* of the user and verify that you can view them in the browser.

Sometimes, mainly for aspect and organization reasons, you do not want to use the notation ~user. To do this you can use an alias.

- Use an *Alias* so that requests that are made to the admin directory are served by the user document you have created. That is, when [*http://localhost/admin /*](http://localhost/admin/) is requested, the *index.html* found in the *public_html* of the user you created is returned.

3.6 Log Files

In these files any error that occurs on the server is recorded. This file is mainly used for debugging *scripts* that hang on the website. If you have to run *scripts,* CGI or PHP the file *error.log* is critical since any error messages that produce the *script* is stored in that file. Apache provides several directives to configure these files.

Log files are also very important when managing the server. The file *access.log* records all accesses to the server so it is very useful for statistical processing.

- Examine the contents of the files. *access.log* and *error.log.* To do this, determine beforehand where they are.

3.7 requests and responses

Finally, we will examine in detail the encapsulation of HTTP connections. To do this:

- Open *Wireshark* and capture its interface. Alternatively, if you don't have wireshark, use your browser's developer tools to examine network requests.
- Make a request via the browser to http://localhost. Examine the request. **Describe the encapsulation. Describe the part of the application level of the request and the server response.**
- Modify the page index.html that you previously created to include the following image <img src=" https://www.upct.es/imagenes/estructura/logo_upct.png">
- Remove the capture and start again.
- Request now for document. Check how many TCP connections are established with your server. **How many TCP connections are established in total? Why?**
- Look for a picture, download it and add it to the document so that you serve it from your server.
- Again request the page and capture exchanges with wireshark**. How many TCP connections are now used?**
- Finally, reload the page and examine the response that gives the server. **What code is received? What does it mean?**