Práctica: Introducción al HTTP

Objetivo

Proporcionar al alumno una visión global de los componentes y procedimientos necesarios en una implementación del protocolo HTTP.

Agente servidor simple python

HTTP es un protocolo basado en la arquitectura cliente servidor que usa TCP como mecanismo de transporte. Prácticamente todos los lenguajes de programación disponen de componentes que implementan las capas de transporte y el nivel sintáctico de HTTP. El lenguaje de programación python implementa de forma nativa un servidor HTTP

- *URL*: https://docs.python.org/2/library/simplehttpserver.html
- *Actividad*: Comprender los mecanismos que participan en la instanciación del agente HTTP servidor que se muestra en el siguiente código.

```
import SimpleHTTPServer
import SocketServer

PORT = 8000
Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("", PORT), Handler)
print "serving at port", PORT
httpd.serve_forever()
```

  - *Preguntas*:
    - ¿Qué componente gestiona el socket de servidor?
    - ¿Qué componente gestiona los mensajes HTTP?
    - ¿Qué método se usa para iniciar el servidor?
- *Actividad*: Copiar el código anterior en un archivo denominado 01-simple-http-server.py y arrancar el agente HTTP.

```
python 01-simple-http-server.py
```

- *Actividad*: Establecer una conexión con el servidor HTTP arrancado en el punto anterior y analizar la estructura del mensaje.

```
telnet localhost 8000
GET / HTTP/1.1
Host:localhost
```

  - *Preguntas:*
    - ¿Qué tipo de mensaje HTTP devuelve el servidor?
    - ¿De qué tipo de contenido es la entidad retornada?
    - ¿Qué tamaño en bytes tiene la entidad retornada en el mensaje HTTP de respuesta?
- *Actividad:* Los servidores web reutilizan las conexiones para conseguir optimización. El cliente decide si la conexión debe permanecer abierta o cerrada a

través de la cabecera *Connection*. En ocasiones los servidores ignoran esta cabecera para evitar un uso excesivo de recursos.

```
telnet locahost 8000
GET / HTTP/1.1
Host:localhost
Connection:keep-alive

GET / HTTP/1.1
Host:localhost
```

Gestión de contenidos

El servidor debe analizar la naturaleza de los recursos que gestiona e informar al cliente a través de un tipo MIME encapsulado en la cabecera *Content-Type*.

- *Actividad*: Crear el fichero hola_mundo.txt con el siguiente contenido:

```
Hola Mundo
```

- *Actividad*: Crear el fichero hola_mundo.html con el siguiente contenido

```
<!DOCTYPE html>
<html>
<head></head>
<body>
	<h1>Hola Mundo</h1>
</body>
</html>
```

- *Actividad*: Crear el fichero hola_mundo_falso.html con el siguiente contenido

```
Hola Mundo
```

- *Actividad*: Crear el fichero hola_mundo_falso.txt con el siguiente contenido

```
<!DOCTYPE html>
<html>
<head></head>
<body>
	<h1>Hola Mundo</h1>
</body>
</html>
```

    - Preguntas:
        - ¿Qué MIME anuncia el servidor para el recurso /hola-mundo.txt?
        - ¿Qué MIME anuncia el servidor para el recurso /hola-mundo.html?
        - ¿Qué MIME anuncia el servidor para el recurso /hola-mundo-falso.txt?
        - ¿Qué MIME anuncia el servidor para el recurso /hola-mundo-falso.html?
        - ¿En general, cuál parece ser la regla para asignar el MIME a un contenido?

- Actividad: Obtener el recurso identificado por el siguiente URL y emplazar copias en los ficheros image.jpg, image, image.png e image.mp4

  http://www.urjc.es/urjcMain/imagenesMain/logoURJC.jpg

  - Preguntas:
    - ¿Qué MIME anuncia el servidor para el recurso /image.jpg?
    - ¿Qué contenido se visualiza en un navegador al descargar el contenido /image.jpg?
    - ¿Qué MIME anuncia el servidor para el recurso /image?
    - ¿Qué contenido se visualiza en un navegador al descargar el contenido /image?
    - ¿Qué MIME anuncia el servidor para el recurso /image.png?
    - ¿Qué contenido se visualiza en un navegador al descargar el contenido /image.png?
    - ¿Qué MIME anuncia el servidor para el recurso /image.mp4?
    - ¿Qué contenido se visualiza en un navegador al descargar el contenido /image.mp4?
- *Actividad*: Obtener el recurso ROOT (/) del agente basado en python
  - Preguntas:
    - ¿Qué contenido retorna el servidor?
    - ¿Qué MIME presenta dicho contenido?

Manejador de peticiones HTTP

El agente HTTP nativo de python permite modificar el comportamiento por defecto e insertar manejadores externos de peticiones HTTP

- *Actividad*: Comprender el flujo de ejecución que tiene lugar cuando se implementa un manejador de mensajes a nivel de aplicación como el que se muestra en el siguiente código

```
import SimpleHTTPServer
import SocketServer

PORT = 8000
class HttpHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        print "REQUESTED RESOURCE: " , self.path

httpd = SocketServer.TCPServer(("", PORT), HttpHandler)
print "serving at port", PORT
httpd.serve_forever()
```

- *Actividad*: Copiar el código anterior en un archivo denominado 02-custom-http-server.py y arrancar el agente HTTP.

```
python 02-custom-http-server.py
```

  - *Preguntas*:
    - ¿Qué contenido retorna el servidor con el nuevo manejador de mensajes?

- *Actividad*: Modificar el manejador de mensajes HTTP para que retorne siempre el código 200 como respuesta al método GET

```
import SimpleHTTPServer
import SocketServer

PORT = 8000
class HttpHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        print "REQUESTED RESOURCE: " , self.path
        self.send_response(200)
        self.send_header('Content-Type','text/html')
        self.end_headers()
        self.wfile.write("<html><body>Tiny server response</body></html>")

httpd = SocketServer.TCPServer(("", PORT), HttpHandler)
print "serving at port", PORT
httpd.serve_forever()
```

- *Actividad*: Modificar el código anterior para que retorne el código 200 para el recurso ROOT (/) y el código 404 para el resto de recursos

```
import SimpleHTTPServer
import SocketServer

PORT = 8000
class HttpHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        print "REQUESTED RESOURCE: " , self.path
        if self.path == "/":
            self.send_response(200)
            self.send_header('Content-Type','text/html')
            self.end_headers()
            self.wfile.write("<html><body>ROOT</body></html>")
        else:
            self.send_error(404,'Resource not found: %s' % self.path)

httpd = SocketServer.TCPServer(("", PORT), HttpHandler)
print "serving at port", PORT
httpd.serve_forever()
```

- *Actividad*: Modificar el manejador HTTP para que retorne el recurso si existe un fichero asociado al path de la petición o el código 404 en caso contrario

```
import SimpleHTTPServer
import SocketServer
from os import curdir, sep, path

PORT = 8000
class HttpHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):
        print "REQUESTED RESOURCE: " , self.path
        fpath = curdir + sep + self.path
        try:
            f = open(fpath)
            fsize = path.getsize(fpath)
        except IOError:
            self.send_error(404,'File Not Found: %s' % self.path)
            return

        self.send_response(200)
```

```
        self.end_headers()
        self.wfile.write(f.read())

httpd = SocketServer.TCPServer(("", PORT), HttpHandler)
print "serving at port", PORT
httpd.serve_forever()
```

- *Preguntas*:
  - ¿Qué valor tiene la cabecera *Content-Type* que retorna el código mostrado arriba?

- *Actividad*: Modificar el manejador HTTP para que retorne el MIME adecuado en base a la extensión del fichero

Servidor multi-dominio

Las servidores HTTP multi-dominio permiten gestionar más de un sitio web a través de un único agente HTTP. Para ello clasifican las peticiones a partir del contenido de la cabecera *Host*.
El servidor Apache es uno de los más usados y usa el concepto de VirtualHost para este propósito. A cada VirtualHost le asigna un directorio separado

```
<VirtualHost *:80>
        ServerName cssai1
        DocumentRoot /Directorio/web/cssai1
 </VirtualHost>

 <VirtualHost *:80>
        ServerName cssai2
         DocumentRoot /Directorio/web/cssai2
 </VirtualHost>
```

- *Actividad*: Modificar el servidor de ficheros desarrollado en la práctica para que soporte múlti-dominio.

```
import SimpleHTTPServer
import SocketServer
from SimpleHTTPServer import SimpleHTTPRequestHandler

PORT = 8000

class HttpMultisiteHandler(SimpleHTTPRequestHandler):

    def do_GET (self):
        host = self.headers.getheader('Host')
        self.path= "/"+host+ self.path
        Handler.do_GET(self)

Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("", PORT), HttpMultisiteHandler)
print "serving at port", PORT
httpd.serve_forever()
```

- *Preguntas*:
  - ¿Es posible usar nombres de sitio de la forma cssai1 ó cssai2?
  - ¿Sería posible acceder con un navegador estándar a un sitio con nombre cssai1 ó cssai2?. ¿Cómo?