Universidad de Costa Rica

Escuela de ciencias de la computación y la informática

Servidor HTTP

Fernando Mata Mora

4 de junio del 2015

Tabla de contenidos

1.Introducción

En el proyecto se nos pidió implementar un servidor HTTP que respondiera Request externos.

1.a.Descripción del problema

El servidor debería ser capaz de responder tres tipos de solicitudes: GET,HEAD y POST de clientes externos siguiendo el protocolo HTTP. Se deberá retornar un código de estado de 202 en caso de que la operación y además se manejaran 2 códigos de retorno en caso de que se intente realizar operaciones que no puedan satisfacer al cliente. En caso de que no se encuentre el archivo se devolverá un código 404 y en caso de que se trate de realizar una operación prohibida por parte del servidor se devolverá un código 406. El servidor deberá ser capaz de procesar los encabezados

2.Análisis del problema

En problema se subdividió en dos grandes partes, que serían la conexión con los clientes y el procesamiento de las solicitudes, en ambos casos se trabaja con una expectativa optimista en la cual se espera que el cliente mande solicitudes HTTP bien formadas.

3.Casos de Prueba

Metodo	Server	URL	Variables	Headers
Pruebas con archivos normales y variables, código esperado 200				
POST	localhost	/logs.html	var1=1&var2=2	
POST	localhost	folder/index.html	var1=1&var2=2	
HEAD	localhost	/logs.html		
HEAD	localhost	/logs.html	var1=1&var2=2	
HEAD	localhost	/folder/index.html	var1=1&var2=2	
GET	localhost	/logs.html		
GET	localhost	/logs.html	var1=1&var2=2	
GET	localhost	/folder/index.html	var1=1&var2=2	
Pruebas con Header Accept diferente del tipo de archivo solicitado, código esperado 406				
POST	localhost	/logs.html		Accept:image/jpeg

HEAD	localhost	/logs.html	var1=1&var2=2	Accept:image/jpeg
GET	localhost	/logs.html	var1=1&var2=2	Accept:image/jpeg
Pruebas con archivos binarios, resultado esperado 200				
POST	localhost	/example.jpg		
HEAD	localhost	/example.jpg		
GET	localhost	/example.jpg		
Pruebas con archivos inexistentes, resultado esperado 404				
POST	localhost	/noexiste.html		
HEAD	localhost	/noexiste.html		
GET	localhost	/noexiste.html		

4.Resultados de los casos de prueba

Los casos de prueba se corrieron usando Postman en todos los casos se obtuvo el resultado esperado de acuerdo a lo descrito anteriormente. Para correr los casos de se incluye en la carpeta del proyecto las solicitudes descritas anteriormente los cuales se pueden correr sin dificultad.

Para correr el script de python nada más es necesario correr el comando “python server.py” y el servidor empezara a escuchar en el puerto 7080.

5.Código fuente con su respectiva documentación interna.

```
import socket
import sys
import os
import xml.etree.ElementTree as ET
import mimetypes
import threading


from datetime import date,time,datetime
import time

#La clase que se encarga de manejar la comunicacion por medio de sockets
class server_interface:
        def __init__(self):
```

```
                self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.client_sock = None
                if self.server_sock is not None:
                        print("Socket created")

#El metodo del servidor por el cual se escuchan a las solicitudes entrantes
        def run(self):
                msg = str()
                self.server_sock.bind(('localhost',7080))
                self.server_sock.listen(5)
                print('Listening on port 7080')
                while 1:
                        (self.client_sock,address) = self.server_sock.accept()
                        if self.client_sock != None:
                                print("Processing entrance")
                                t =
threading.Thread(target=client_thread,args=(self.client_sock,))
                                t.start()
                                self.client_sock = None

#El thread que corre el server por cada una de las solicitudes
def client_thread(socket):
        request_handler = Request_Handler()
        msg = socket.recv(2048)
        response = request_handler.process_request(msg.decode("utf-8"))
        socket.sendall(response)
        socket.close()

#La clase que se encarga de manejar los request que llegan al API
class Request_Handler:
        def __init__(self):
                self.request_parts = dict()
                self.response_parts = dict()
                self.response_parts["response_headers"] = list()
                self.response_parts["response_body"] = ""

#El metodo que se encarga de procesar el request que se le hizo al servidor
        def process_request(self,request):
                self.process_request_parts(request)
                self.add_to_log()
                if self.valid_request():
                        if self.request_parts["request_type"] == "GET":
                                self.retrive_method()

                        if self.request_parts["request_type"] == "HEAD":
                                self.retrive_method()
```

```
                        self.response_parts["response_body"]= ""

                    if self.request_parts["request_type"] == "POST":
                        self.retrive_method()
                else:
                    self.response_parts["http_response_code"] = "HTTP/1.1 406 NOT
ACCEPTABLE"
                    self.response_parts["response_body"] = ""

                self.other_headers()
                return self.make_response()

#El metodo que se encarga de buscar el archivo solicitado al servidor para despues
devolverlo
        def retrive_method(self):
                try:
                    path = "." + self.request_parts["request_doc"]
                    print(os.path.abspath(path))
                    file_name = os.path.basename(path)

                    (file_type, file_encoding) =
mimetypes.guess_type(self.request_parts["request_doc"])
                    print("Tipo de archivo: "+file_type)
                    if "text" in file_type:
                        self.response_parts["response_body"] =
open(os.path.abspath(path),'r').read()
                    else:
                        self.response_parts["response_body"] =
open(os.path.abspath(path),'rb').read()
                    self.response_parts["http_response_code"] = "HTTP/1.1 200 OK"
                    self.response_parts["response_headers"].append("Content-Length:" +
str(len(self.response_parts["response_body"])))
                    self.response_parts["response_headers"].append("Content-Type:" +
file_type)

                except Exception as e:
                    print(e)
                    self.response_parts["http_response_code"] = "HTTP/1.1 404 NOT
FOUND"

#Metodo que se encarga de validar que el tipo de datos que se piden son iguales al tipo de
datos del documento solicitado
        def valid_request(self):
                mimetypes.init()
                print(mimetypes.guess_type(self.request_parts["request_doc"])[0])
                print( self.request_parts["request_headers"]["Accept"])
```

```
                if self.request_parts["request_headers"]["Accept"] is not None:
                        if "*/*" not in self.request_parts["request_headers"]["Accept"]:
                                if mimetypes.guess_type(self.request_parts["request_doc"])[0]
!= self.request_parts["request_headers"]["Accept"] :
                                        return False
                return True

#Metodo que se encarga de identificar las diferentes partes del request para procesarlo y
generar un response
        def process_request_parts(self,request_text):
                self.request_parts = dict()
                request_chunks = request_text.split('\r\n')
                request_item = request_chunks[0].split()
                request_headers = request_chunks[1:]
                self.request_parts['request_type'] = request_item[0]
                self.request_parts["request_doc"] = request_item[1]
                self.request_parts["request_variables"] = ""
                self.request_parts["request_headers"] = ""
                if self.request_parts["request_type"] == "GET" or
self.request_parts["request_type"] == "HEAD":
                        if "?" in self.request_parts["request_doc"]:
                                doc_and_data = self.request_parts["request_doc"].split("?")
                                self.request_parts["request_doc"] = doc_and_data[0]
                                self.request_parts["request_variables"] = doc_and_data[1]
                if self.request_parts["request_type"] == "POST":

self.request_parts["request_variables"]=request_headers[len(request_headers)-1]
                        request_headers.pop(len(request_headers)-1)
                self.request_parts["request_headers"] =
self.process_headers(request_headers)
                print(self.request_parts)

#Metodo que se encarga de procesar los diferentes headers existentes para su
identificacion
        def process_headers(self,request_headers):
                headers_dict = dict()
                for header in request_headers:
                        header_parts = header.split(":")
                        if len(header_parts) > 1:
                                headers_dict[header_parts[0]] = header_parts[1]
                return headers_dict
#Metodo que agrega a la bitacora cada solicitud hecha al servidor
        def add_to_log(self):
                tree = ET.parse("log.html")
                table = tree.findall("./body/table")[0]
                row = ET.SubElement(table,'tr')
```

```
                ET.SubElement(row,'td').text = self.request_parts["request_type"]
                ET.SubElement(row,'td').text = str(int(time.time()))
                ET.SubElement(row,'td').text = 'localhost'
                ET.SubElement(row,'td').text = 'localhost'
                ET.SubElement(row,'td').text = self.request_parts["request_doc"]
                ET.SubElement(row,'td').text = self.request_parts["request_variables"]
                tree.write("log.html")
#Metodo que agrega los otros headers necesarios para la respuesta
        def other_headers(self):
                self.response_parts["response_headers"].append("Date:" +
datetime.now().strftime("%a, %d %b %Y %H:%M:%S %Z"))
                self.response_parts["response_headers"].append("Server: Proyecto App Web
Servidor")
#Metodo que se encarga de unir las diferentes partes de la respuesta para pasarla al cliente
        def make_response(self):
                response= bytes()
                response +=
self.response_parts["http_response_code"].encode()+"\r\n".encode()
                for e in self.response_parts["response_headers"]:
                        response += e.encode() +"\r\n".encode()
                response += "\r\n".encode()
                if self.response_parts["response_body"] != "":
                        if type(self.response_parts["response_body"]) is str:
                                response += (self.response_parts["response_body"]).encode()
                        else:
                                response += (self.response_parts["response_body"])
                return response

def main():
        server = server_interface()
        server.run()

if __name__ == "__main__":
        main()
```

6.Bibliografía

[Protocolo HTTP]
[HTTP Headers]
[Python OS library]
[Python Sockets]