# MEMORIA P1.3

Monterroso Barco, Alberto

Código fuente: <https://github.com/Albermonte/LRSS/tree/master/P1.3>

Se ha dividido la práctica en varios archivos para que sea más sencillo reutilizar código y más limpio a la hora de programar. Gracias a esto se ha hecho una versión simple (main.py) y una completa (main\_select.py)

# main.py

import sys

import signal

import socket

import re

from utils.res import Res

if len(sys.argv) < 2:

    print("Missing param PORT.\n")

    quit()

RECV\_BUFFER = 1024

PORT = int(sys.argv[1])

print(f"Running server on Port: {PORT}")

# Create socket

# Listen for new clients

# Send array of user info with connections to clients

print("Creating Socket")

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

def sig\_handler(signum, frame):

    print("\nClosing socket...")

    sock.close()

    quit()

signal.signal(signal.SIGINT, sig\_handler)

# Reuse address, no more address already in use error

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

print("Binding address and port")

server\_address = ('0.0.0.0', PORT)

sock.bind(server\_address)

print("Listening...")

sock.listen()

while True:

    conn, addr = sock.accept()

    req = conn.recv(RECV\_BUFFER).decode()

    res = Res(conn)

    # Check if request is valid

    result = re.search('GET (.\*) HTTP/', req)

    if not result:

        res.not\_found()

        conn.close()

        continue

    # Get requested file and redirect to index.html if no file requested

    req\_file = result.group(1)

    if req\_file == "/":

        req\_file = "/index.html"

    # Get the file from public folder

    filename = "./public" + req\_file

    res.send(filename)

    conn.close()

# Source:

#   https://gist.github.com/joncardasis/cc67cfb160fa61a0457d6951eff2aeae

#   https://iximiuz.com/en/posts/writing-web-server-in-python-sockets/

#   https://www.codementor.io/@joaojonesventura/building-a-basic-http-server-from-scratch-in-python-1cedkg0842

#   https://medium.com/geekculture/implementing-http-from-socket-89d20a1f8f43

# main\_select.py

import sys

import signal

import socket

import select

import re

from utils.res import Res

if len(sys.argv) < 2:

    print("Missing param PORT.\n")

    quit()

RECV\_BUFFER = 1024

PORT = int(sys.argv[1])

# Timeout in seconds

TIMEOUT = 10

print(f"Running server on Port: {PORT}")

# Create socket

# Listen for new clients

# Send array of user info with connections to clients

print("Creating Socket")

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

def sig\_handler(signum, frame):

    print("\nClosing socket...")

    sock.close()

    quit()

signal.signal(signal.SIGINT, sig\_handler)

# Reuse address, no more address already in use error

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

print("Binding address and port")

server\_address = ('0.0.0.0', PORT)

sock.bind(server\_address)

print("Listening...")

sock.listen()

is\_timeout = False

# List of sockets for select.select()

sockets\_list = [sock]

while True:

    read\_sockets, \_, exception\_sockets = select.select(

        sockets\_list, [], sockets\_list, TIMEOUT)

    notified\_socket: socket.socket

    for notified\_socket in read\_sockets:

        if notified\_socket == sock:

            # Accept new connection

            client\_socket, client\_address = sock.accept()

            # Add accepted socket to select.select() list

            sockets\_list.append(client\_socket)

        else:

            # We are active so no restart timeout

            is\_timeout = False

            req = notified\_socket.recv(RECV\_BUFFER).decode()

            # Check if keep-alive header is present

            keep\_alive = re.search('Connection: keep-alive', req)

            res = Res(notified\_socket, "HTTP/1.1", not not keep\_alive)

            # Check if request is valid

            result = re.search('(GET|HEAD)(.\*) HTTP/', req)

            if not result:

                res.not\_found()

                notified\_socket.close()

                sockets\_list.remove(notified\_socket)

                continue

            # Remove first space from regex result

            req\_file = result.group(2).lstrip()

            # Get requested file and redirect to index.html if no file requested

            if req\_file == "/":

                req\_file = "/index.html"

            # If not found, send 404

            elif not req\_file:

                # Default 404 file if none present in public folder

                req\_file = "/404.html"

            filename = "./public" + req\_file

            # Check if request method is HEAD and send only the headers

            head\_only = result.group(1) == "HEAD"

            r = res.send(filename, not not head\_only)

            # If keep-alive header is not present, close the socket

            # If send returned -1 (file not found), close the socket

            if not keep\_alive or r == -1:

                notified\_socket.close()

                sockets\_list.remove(notified\_socket)

    for notified\_socket in exception\_sockets:

        # Remove from list for socket.socket()

        sockets\_list.remove(notified\_socket)

    if not (read\_sockets or exception\_sockets):

        if not is\_timeout:

            print("Servidor web inactivo")

            # Timeout to true to not send the innactive message every 10 seconds

            is\_timeout = True

            for notified\_socket in sockets\_list:

                if notified\_socket != sock:

                    notified\_socket.close()

                    sockets\_list.remove(notified\_socket)

# Source:

#   https://gist.github.com/joncardasis/cc67cfb160fa61a0457d6951eff2aeae

#   https://iximiuz.com/en/posts/writing-web-server-in-python-sockets/

#   https://www.codementor.io/@joaojonesventura/building-a-basic-http-server-from-scratch-in-python-1cedkg0842

#   https://medium.com/geekculture/implementing-http-from-socket-89d20a1f8f43

# utils/res.py

import os

import sys

import datetime

import re

from socket import socket

# Class to manage response

class Res:

    conn: socket

    # All regex to check file type

    regex\_html = re.compile(r"html")

    regex\_image = re.compile(r"gif|jpe?g|bmp|png")

    regex\_css = re.compile(r"css")

    regex\_js = re.compile(r"js")

    regex\_json = re.compile(r"json")

    regex\_xml = re.compile(r"xml")

    http\_ver = "HTTP/1.0"

    headers = f"{http\_ver} 200 OK\nServer: LRSS/1.0.0\n"

    def \_\_init\_\_(self, conn: socket, ver = "HTTP/1.0", keep\_alive = False):

        self.conn = conn

        self.http\_ver = ver

        self.keep\_alive = keep\_alive

        # print(f"New connection from {conn.getpeername()}, version: {self.http\_ver}, keep alive: {self.keep\_alive}")

    def send(self, file, head\_only = False):

        # Check if file exists

        if os.path.exists(file):

            self.headers += f"Date: {datetime.datetime.now()}\n"

            # Check file type

            if re.findall(self.regex\_html, file):

                self.headers += f"Content-type: text/html\n"

            elif re.findall(self.regex\_image, file):

                type = re.findall(self.regex\_image, file)[0]

                self.headers += f"Content-type: image/{type}\n"

            elif re.findall(self.regex\_css, file):

                self.headers += f"Content-type: text/css\n"

            elif re.findall(self.regex\_js, file):

                self.headers += f"Content-type: text/javascript\n"

            elif re.findall(self.regex\_json, file):

                self.headers += f"Content-type: application/json\n"

            elif re.findall(self.regex\_xml, file):

                self.headers += f"Content-type: application/xml\n"

            # Set lenght header

            # Set connection header, important for keep-alive

            self.headers += f"Content-length: {os.path.getsize(file)}\nConnection: {'keep-alive' if self.keep\_alive else 'close'}\n\n"

            self.conn.sendall(bytes(self.headers, "utf-8"))

            # If request method is HEAD, don't send body

            if not head\_only:

                self.conn.sendall(bytes(open(file, "rb").read()))

        else:

            return self.not\_found()

        return 0

    def not\_found(self):

        # Send 404 error

        body = "<html><body>404 Not Found</body></html>"

        headers = f"{self.http\_ver} 404 Not Found\nDate: {datetime.datetime.now()}\nServer: LRSS/1.0.0\nContent-type: text/html\nContent-length: {sys.getsizeof(body)}\nConnection: close\n\n"

        self.conn.sendall(bytes(headers + body, "utf-8"))

        return -1

    # Test method in case it's needed

    def test(self):

        body = "<html><body>Hello World</body></html>"

        headers = f"{self.http\_ver} 200 OK\nDate: {datetime.datetime.now()}\nServer: LRSS/1.0.0\nContent-type: text/html\nContent-length: {sys.getsizeof(body)}\nConnection: close\n\n"

        self.conn.sendall(bytes(headers + body, "utf-8"))

        return 0

También se ha desarrollado un pequeño programa para testear la diferencia entre las versiones, simplemente hace request a una url en la que se devuelve un mensaje de 404, para el cual no es necesario acceder al disco para evitar diferencias de tiempo producidas por la lectura del archivo. Estas request se hacen en paralelo usando threads y se contabiliza el tiempo total que ha tomado, viendo que en la versión completa, al usar select, se atienden a más request y el programa tarda menos

# test.py

import requests

import time

from concurrent.futures import ThreadPoolExecutor

CONNECTIONS = 5000

list\_of\_urls = []

for i in range(CONNECTIONS):

    list\_of\_urls.append("http://localhost:3000/test")

def get\_url(url):

    return requests.get(url)

time1 = time.time()

with ThreadPoolExecutor(max\_workers=CONNECTIONS) as pool:

    pool.map(get\_url, list\_of\_urls)

time2 = time.time()

print(f'Took {time2-time1:.2f} s')

Primera prueba ejecutada usando main.py y segunda main\_select.py

