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)

```python
# 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
```

```python
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

```
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.3$ python3 test.py
Took 7.15 s
albermonte@DESKTOP-HEU9CTS:/mnt/c/Users/Alberto/OneDrive - Universidad de Alcala/LRSS/LRSS/P1.3$ python3 test.py
Took 7.01 s
```