Práctica 1.3 Laboratorio de Redes, Sistemas y Servicios

Pedro Gómez Martín y Mario Aguilera Alcalde

5 de mayo de 2021

1. server.py

```python
#!/usr/bin/env python3

from select import select
from socket import AF_INET, SOCK_STREAM, socket
from sys import argv, exit, stderr
from petition import Petition
from http_methods import handle_get, handle_delete, handle_post, register_functions
from signal import signal, SIGINT


def setup() -> socket:
    """
    Create a socket for the server to listen
    """
    # Create a TCP/IP socket
    server = socket(AF_INET, SOCK_STREAM)
    server.setblocking(False)

    def sigint_handler(sig, frame):
        """
        Catches a SIGINT and cleans up
        """
        print("[i] Caught SIGINT, cleaning up...")
        server.close()
        exit(0)

    signal(SIGINT, sigint_handler)

    # Parse arguments
    if len(argv) != 2:
        print(f"Usage\n\t{argv[0]} <port>")
        exit(1)

    try:
        server_address = ('', int(argv[1]))
        print(f'starting up on port {server_address[1]}', file=stderr)
        server.bind(server_address)
    except ValueError as e:
        print(f"Error while trying to parse arguments {e}")
        exit(1)
    except OSError as e:
        print(f"Error while trying to bind to {argv[1]}: {e.strerror}")
        exit(1)
```

```
    # Listen for incoming connections
    server.listen(5)

    register_functions()

    return server


def main():
    server = setup()

    # Sockets from which we expect to read
    inputs = [server]

    # This loop blocks until there is data ready
    while inputs:
        # Wait for at least one of the sockets to be ready for processing
        readable, writable, exceptional = select(inputs, [], [], 10)
        if len(readable) == 0:
            print("[i] Servidor inactivo")

        # Handle inputs
        for s in readable:
            # A "readable" listening socket is ready to accept a connection
            if s is server:
                handle_new(s, inputs)

            # A message incoming from a client
            else:
                handle_msg(s, inputs, server)


def handle_new(s: socket, inputs: list[socket]):
    """
    Handles an incoming connection from a client
    """
    connection, client_address = s.accept()
    print('  connection from', client_address, file=stderr)
    connection.setblocking(False)
    inputs.append(connection)


def handle_msg(s: socket, inputs: list[socket], server: socket):
    """
    Handles a msg from an incomming connection, parses the request into a class
    and then forwards the request into the corresponding handler
    """
    buff = bytes()
    # Put the socket in a list to pass it to select with timeout
    s_list = [s]

    while True:
        readable, writable, exceptional = select(s_list, [], [], 10)
        if len(readable) == 0:
            print("[x] Invalid petition")
            return
```

```
            data = s.recv(1024)

            if len(data) == 0:
                return

            buff += data
            slice_obj = slice(-1, -5, -1)

            last_chars = buff[slice_obj]

            if last_chars.decode() == "\n\r\n\r":
                break

        petition = Petition(data.decode())

        if len(petition.method) != 0:
            if petition.method == 'GET':
                handle_get(s, petition)

            elif petition.method == 'POST':
                handle_post(s, petition)

            elif petition.method == 'DELETE':
                handle_delete(s, petition)

        if not petition.keep_alive:
            addr, port = s.getpeername()
            print(f'  closing {addr}:{port}', file=stderr)
            # Stop listening for input on the connection
            inputs.remove(s)
            s.close()


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

2. http methods.py

```
from petition import Petition
from datetime import date
from socket import socket
from typing import Dict, Callable
from datetime import datetime
import mimetypes


router: Dict[str, Callable[[socket, Petition], None]] = {}


def handle_log_query(s: socket, p: Petition):
    """
    Simply logs the request and returns OK
    """
    data = "OK".encode()

    print(f"[i] Got {p.header_map}")
```

```
    resp = craft_response("200 OK", "text/plain", data)
    s.sendall(resp)


def handle_time_query(s: socket, p: Petition):
    """
    Returns the current date and time in the utc timezone
    """
    data = str(datetime.utcnow()).encode()

    resp = craft_response("200 OK", "text/plain", data)
    s.sendall(resp)


def register_functions():
    """
    Registers the log and time endpoints
    """
    router["/api/time"] = handle_time_query
    router["/api/log"] = handle_log_query


def handle_post(s: socket, petition: Petition):
    pass


def handle_delete(s: socket, petition: Petition):
    pass


def handle_get(s: socket, petition: Petition):
    """
    Handles a get petition
    """
    path = petition.arguments[0]

    try:
        route_handler = router[path]
        route_handler(s, petition)
        return
    except KeyError:
        pass

    if path[-1] == '/':
        path += "index.html"

    mime, _ = mimetypes.guess_type(path)

    if mime is None:
        mime = "text/plain"

    print(f"[i] GET for \"{path}\" with {petition.header_map}")

    try:
        with open("./" + path, "rb") as file:
            data = file.read()
            resp = craft_response("200 OK", mime, data)
```

```
    except FileNotFoundError:
        resp = craft_response("404 NOT FOUND", mime, ''.encode())

    s.setblocking(True)
    s.sendall(resp)
    s.setblocking(False)


def craft_response(status: str, mime: str, data: bytes) -> bytes:
    header_status = f"HTTP/1.1 {status}\r\n"
    header_date = f"Date: {str(date.today())}\r\n"
    header_server = "Server: Anarres\r\n"
    header_mime = f"Content-type: {mime}\r\n"
    header_length = f"Content-length: {len(data)}\r\n"
    header_connection = f"Connection: {'close'}\r\n\r\n"

    response = header_status + header_date + header_server + header_mime + header_length + header_co

    rsp = response.encode() + data

    return rsp
```

3. petition.py

```
from typing import List


class Petition:
    """
    Parses a petition string into this object
    """
    method: str
    arguments: List[str]
    keep_alive: bool

    def __init__(self, data: str):
        headers = data.split('\r\n')
        headers = list(filter(lambda x: x.strip() != '', headers))
        header_map = {}

        method = headers.pop(0).split(' ')

        self.method = method.pop(0)
        self.arguments = method

        for header in headers:
            kv = header.split(' ')
            key = kv.pop(0)
            key = list(key)
            key.pop(-1)
            key = "".join(key)

            header_map[key] = kv

        self.header_map = header_map
```

```
try:
    header_map['Keep-Alive']
    self.keep_alive = True
except KeyError:
    self.keep_alive = False
```