# Memoria P2

Monterroso Barco, Alberto

Código fuente: <https://github.com/Albermonte/LRSS/tree/master/P1.2>

## Cliente-Servidor

# servidor.py

import sys

import signal

import socket

import select

import json

if len(sys.argv) < 2:

    print("Missing param PORT.\n")

    quit()

PORT = int(sys.argv[1])

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

RECV\_BUFFER = 1024

# Socket TCP

# Conect non-blockin

# Listen for msgs from every client

# Send msg to every client except the origin

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 = ('localhost', PORT)

sock.bind(server\_address)

print("Listening...")

sock.listen()

# List of sockets for select.select()

sockets\_list = [sock]

# List of clients

client\_list = {}

def receive\_message(client\_socket: socket.socket):

    try:

        # data = {

        #     "username": "",

        #     "message": ""

        # }

        data = client\_socket.recv(RECV\_BUFFER)

        # If we received no data, client gracefully closed a connection, for example using socket.close() or socket.shutdown(socket.SHUT\_RDWR)

        if not len(data):

            return False

        data = data.decode('utf-8')

        print(f"Message data: {data}")

        if not data.startswith("file-"):

            data = json.loads(data)

            return data

        else:

            filename = data.split("file-")[1] + ".temp"

            # It's a file then, could be improved because here we will catch other errors :/

            with open(filename, "wb") as f:

                while True:

                    # read 1024 bytes from the socket (receive)

                    bytes\_read = client\_socket.recv(RECV\_BUFFER)

                    if not bytes\_read:

                        # TODO: Not working, not reaching here

                        print("File received")

                        break

                    # write to the file the bytes we just received

                    f.write(bytes\_read)

            # TODO: Not working, not reaching here

            print(f"Filename: {filename}")

            return filename

    except Exception as e:

        print("Error receiving msg")

        print(e)

        # Some error or disconection

        return False

while True:

    # Calls Unix select() system call or Windows select() WinSock call with three parameters:

    #   - rlist - sockets to be monitored for incoming data

    #   - wlist - sockets for data to be send to (checks if for example buffers are not full and socket is ready to send some data)

    #   - xlist - sockets to be monitored for exceptions (we want to monitor all sockets for errors, so we can use rlist)

    # Returns lists:

    #   - reading - sockets we received some data on (that way we don't have to check sockets manually)

    #   - writing - sockets ready for data to be send thru them

    #   - errors  - sockets with some exceptions

    # This is a blocking call, code execution will "wait" here and "get" notified in case any action should be taken

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

        sockets\_list, [], sockets\_list)

    # Iterate over notified sockets

    for notified\_socket in read\_sockets:

        # If notified socket is a server socket - new connection, accept it

        if notified\_socket == sock:

            # Accept new connection

            # That gives us the client socket and the ip/port

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

            # The next message is the client username with the connecting message

            user = receive\_message(client\_socket)

            # If False - client disconnected before he sent his name

            if user is False:

                continue

            # Add accepted socket to select.select() list

            sockets\_list.append(client\_socket)

            # Also save user

            client\_list[client\_socket] = user

            print(

                f"Accepted new connection from {client\_address} with username: {user['username']}")

            # Feature: Send message to all clients about new client connected

            client\_socket: socket.socket

            for client\_socket in client\_list:

                data = {

                    "username": user['username'],

                    "message": "Entered the chat!"

                }

                data = json.dumps(data)

                data = bytes(data, "utf-8")

                client\_socket.send(data)

        # Else existing socket is sending a message

        else:

            # Receive message

            message = receive\_message(notified\_socket)

            print(f"Message: {message}")

            # If False, client disconnected, cleanup

            if message is False:

                print(

                    f"Closed connection from: {client\_list[notified\_socket]['username']}")

                # Feature: Send message to all clients about client disconnected

                client\_socket: socket.socket

                for client\_socket in client\_list:

                    # But don't sent it to sender

                    if client\_socket != notified\_socket:

                        data = {

                            "username": user['username'],

                            "message": "Left the chat!"

                        }

                        data = json.dumps(data)

                        data = bytes(data, "utf-8")

                        client\_socket.send(data)

                # Remove from list for socket.socket()

                sockets\_list.remove(notified\_socket)

                # Remove from our list of users

                del client\_list[notified\_socket]

                continue

            # Get user by notified socket, so we will know who sent the message

            user = client\_list[notified\_socket]

            if not type(message) is dict:

                print(

                    f"Received message from {user['username']} : {message['message']}")

            # Iterate over connected clients and broadcast message

            client\_socket: socket.socket

            for client\_socket in client\_list:

                # But don't sent it to sender

                if client\_socket != notified\_socket:

                    if not type(message) is dict:

                        sock.send(bytes(message, "utf-8"))

                        with open(message, "rb") as f:

                            print(f"Sending file {message}")

                            while True:

                                # Read the bytes from the file

                                bytes\_read = f.read(RECV\_BUFFER)

                                if not bytes\_read:

                                    # file transmitting is done

                                    print("File sent")

                                    break

                                sock.sendall(bytes\_read)

                        continue

                    data = {

                        "username": user['username'],

                        "message": message['message']

                    }

                    data = json.dumps(data)

                    data = bytes(data, "utf-8")

                    client\_socket.send(data)

    # It's not really necessary to have this, but will handle some socket exceptions just in case

    for notified\_socket in exception\_sockets:

        # Remove from list for socket.socket()

        sockets\_list.remove(notified\_socket)

        # Remove from our list of users

        del client\_list[notified\_socket]

# Sources:

# https://pythonprogramming.net/server-chatroom-sockets-tutorial-python-3/

# https://mirdan.medium.com/send-json-with-python-socket-f1107876f50e

# cliente.py

import errno

import signal

import sys

import select

import socket

import json

from pathlib import Path

def delete\_last\_line():

    # Delete last line from stdout

    sys.stdout.write('\x1b[2K')

def is\_file(path):

    return Path(path).exists()

if len(sys.argv) < 3:

    print("Missing params.\n")

    quit()

if not sys.argv[2].isnumeric():

    print(

        f"Port \"{sys.argv[2]}\" not numeric, usage: python3 ping\_oc.py host port\n")

    quit()

HOST = sys.argv[1]

PORT = int(sys.argv[2])

RECV\_BUFFER = 1024

print(f"Running client on {HOST}:{PORT}\n")

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

sock.connect((HOST, PORT))

# Set recv to not blocking so we can do things while waiting for msg

sock.setblocking(False)

def sig\_handler(signum, frame):

    print("\nClosing socket...")

    sock.close()

    quit()

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

# Ask user for username

username = input("Enter your username: ")

if not username:

    username = "Anonymous"

print(f"You choosed {username} as username \n\n")

# First message for server

data = {

    "username": username,

    "message": "connecting"

}

# Convert to json and send

data\_send = json.dumps(data)

data\_send = bytes(data\_send, "utf-8")

sock.send(data\_send)

print("###### Connected ######\n\n")

# flush=True to avoid errors, without it this line was not printed

print("You > ", end="", flush=True)

while True:

    # Feature: Non blocking input, receive messages while typing

    is\_input, \_, \_ = select.select([sys.stdin], [], [], 0)

    if is\_input:

        message = sys.stdin.readline().strip()

        # If not message (eg: \n) don't send it

        if message:

            if is\_file(message):

                sock.send(bytes(f"file-{message}", "utf-8"))

                with open(message, "rb") as f:

                    print(f"Sending file {message}")

                    while True:

                        # Read the bytes from the file

                        bytes\_read = f.read(RECV\_BUFFER)

                        if not bytes\_read:

                            # file transmitting is done

                            print("File sent")

                            break

                        sock.sendall(bytes\_read)

            else:

                # TODO: Check if message + username + data > 1024

                data["message"] = message

                # print(f"Sending {data}")

                data\_send = json.dumps(data)

                data\_send = bytes(data\_send, "utf-8")

                sock.send(data\_send)

                print("You > ", end="", flush=True)

    try:

        while True:

            data\_received = sock.recv(RECV\_BUFFER)

            # The server was closed

            if not len(data\_received):

                print("Connection lost")

                sig\_handler(0, 0)

            # Convert string to json

            data\_received = data\_received.decode('utf-8')

            data\_received = json.loads(data\_received)

            # Delete last line and print data, this will replace "You >" with another client message

            delete\_last\_line()

            print(f"{data\_received['username']} : {data\_received['message']}")

            print("You > ", end="", flush=True)

    except IOError as e:

        # This is normal on non blocking connections - when there are no incoming data error is going to be raised

        if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:

            print(f"Reading error: {str(e)}")

            sys.exit()

        # We just did not receive anything

        continue

    except Exception as e:

        # Any other exception - something happened, exit

        print(f"Reading error: {str(e)}")

        sig\_handler(0, 0)

# Sources:

#  https://repolinux.wordpress.com/2012/10/09/non-blocking-read-from-stdin-in-python/

#  https://pythonprogramming.net/client-chatroom-sockets-tutorial-python-3/?completed=/server-chatroom-sockets-tutorial-python-3/

#  https://stackoverflow.com/questions/21791621/taking-input-from-sys-stdin-non-blocking

#  https://www.thepythoncode.com/article/send-receive-files-using-sockets-python

## P2P

# servidor\_usuarios.py

import sys

import signal

import socket

import select

import json

if len(sys.argv) < 2:

    print("Missing param PORT.\n")

    quit()

PORT = int(sys.argv[1])

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

RECV\_BUFFER = 1024

# 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 = ('localhost', PORT)

sock.bind(server\_address)

print("Listening...")

sock.listen()

# List of sockets for select.select()

sockets\_list = [sock]

# List of clients

client\_list = {}

client\_connections\_list = {}

def receive\_message(client\_socket: socket.socket):

    try:

        # data = {

        #     "username": "",

        #     "message": ""

        # }

        data = client\_socket.recv(RECV\_BUFFER)

        # If we received no data, client gracefully closed a connection, for example using socket.close() or socket.shutdown(socket.SHUT\_RDWR)

        if not len(data):

            return False

        data = data.decode('utf-8')

        print(f"Message data: {data}")

        data = json.loads(data)

        return data

    except:

        # Some error or disconection

        return False

while True:

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

        sockets\_list, [], sockets\_list)

    # Iterate over notified sockets

    for notified\_socket in read\_sockets:

        if notified\_socket == sock:

            # Accept new connection

            # That gives us the client socket and the ip/port

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

            # The next message is the client username with the connecting message

            user = receive\_message(client\_socket)

            # If False - client disconnected before he sent his name

            if user is False:

                continue

            # Add accepted socket to select.select() list

            sockets\_list.append(client\_socket)

            # Also save user

            client\_list[client\_socket] = user

            # Save connection

            client\_connections\_list[user["username"]] = (

                client\_socket.getpeername()[0], user["port"])

            print(

                f"Accepted new connection from {client\_address} with username: {user['username']}")

            client\_socket: socket.socket

            for client\_socket in client\_list:

                # Send list of servers

                client\_connections\_list\_serialized = json.dumps(

                    client\_connections\_list)

                client\_connections\_list\_serialized = bytes(

                    client\_connections\_list\_serialized, "utf-8")

                client\_socket.send(client\_connections\_list\_serialized)

        else:

            # Receive message

            message = receive\_message(notified\_socket)

            # If False, client disconnected, cleanup

            if message is False:

                print(

                    f"Closed connection from: {client\_list[notified\_socket]['username']}")

                # Feature: Send message to all clients about client disconnected

                client\_socket: socket.socket

                for client\_socket in client\_list:

                    # But don't sent it to sender

                    if client\_socket != notified\_socket:

                        data = {

                            "username": user['username'],

                            "message": "Left the chat!"

                        }

                        data = json.dumps(data)

                        data = bytes(data, "utf-8")

                        client\_socket.send(data)

                # Remove from client\_connection\_list

                try:

                    del client\_connections\_list[user["username"]]

                except:

                    # Nothing on the list

                    client\_connections\_list = {}

                    pass

                # Remove from list for socket.socket()

                sockets\_list.remove(notified\_socket)

                # Remove from our list of users

                del client\_list[notified\_socket]

                continue

    # It's not really necessary to have this, but will handle some socket exceptions just in case

    for notified\_socket in exception\_sockets:

        # Remove from list for socket.socket()

        sockets\_list.remove(notified\_socket)

        # Remove from our list of users

        del client\_list[notified\_socket]

# peer.py

import errno

import signal

import sys

import select

import socket

import json

def delete\_last\_line():

    # Delete last line from stdout

    sys.stdout.write('\x1b[2K')

if len(sys.argv) < 3:

    print("Missing params.\n")

    quit()

if not sys.argv[2].isnumeric():

    print(

        f"Port \"{sys.argv[2]}\" not numeric, usage: python3 ping\_oc.py host port\n")

    quit()

HOST = sys.argv[1]

PORT = int(sys.argv[2])

RECV\_BUFFER = 1024

print(f"Running client on {HOST}:{PORT}\n")

# Connect to server

# Receive list of clients

# Connect to every client

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

sock.connect((HOST, PORT))

# Set recv to not blocking so we can do things while waiting for msg

# sock.setblocking(False)

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

# Assigning free port https://stackoverflow.com/a/1365284/7312697

sock\_server.bind(("", 0))

# Reuse address, no more address already in use error

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

# sock\_server.setblocking(False)

sock\_server.listen(50)

print(f"Socket server on: {sock\_server.getsockname()}")

def sig\_handler(signum, frame):

    print("\nClosing socket...")

    sock.close()

    quit()

def delete\_last\_line():

    # Delete last line from stdout

    sys.stdout.write('\x1b[2K')

def receive\_message(client\_socket: socket.socket):

    try:

        data = client\_socket.recv(RECV\_BUFFER)

        # If we received no data, client gracefully closed a connection, for example using socket.close() or socket.shutdown(socket.SHUT\_RDWR)

        if not len(data):

            return False

        data = data.decode('utf-8')

        data = json.loads(data)

        return data

    except Exception as e:

        print("Error receiving msg")

        print(e)

        # Some error or disconection

        return False

def connect\_to\_peers(sock: socket.socket):

    data\_received = sock.recv(RECV\_BUFFER)

    # The server was closed

    if not len(data\_received):

        print("Connection lost")

        sig\_handler(0, 0)

    # Convert string to json

    data\_received = data\_received.decode('utf-8')

    data\_received = json.loads(data\_received)

    # Deleting our peer

    if data\_received[username]:

        del data\_received[username]

    client\_connections\_list = data\_received

    print(client\_connections\_list)

    for client\_name in client\_connections\_list:

        conn = client\_connections\_list[client\_name]

        ip = conn[0]

        port = conn[1]

        print(f"Connecting to {ip}:{port}")

        new\_socket = socket.socket(

            socket.AF\_INET, socket.SOCK\_STREAM)

        try:

            new\_socket.connect((ip, port))

            sockets\_list.append(new\_socket)

        except Exception as e:

            print(f"Error connecting to {ip}:{port}")

            print(str(e))

            new\_socket.close()

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

# Ask user for username

username = input("Enter your username: ")

if not username:

    username = "Anonymous"

print(f"You choosed {username} as username \n\n")

# First message for server

data = {

    "username": username,

    "port": sock\_server.getsockname()[1]

}

# Convert to json and send

data\_send = json.dumps(data)

data\_send = bytes(data\_send, "utf-8")

sock.send(data\_send)

sockets\_list = [sock\_server, sys.stdin]

client\_connections\_list = {}

print("Connecting to peers")

connect\_to\_peers(sock)

print("###### Connected ######\n\n")

print("You > ", end="", flush=True)

while True:

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

        sockets\_list, [], [])

    notified\_socket: socket.socket

    for notified\_socket in read\_sockets:

        if notified\_socket == sock\_server:

            # Some client is sending a message

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

            # Receive message

            sockets\_list.append(client\_socket)

            delete\_last\_line()

            print(f"New peer connected {client\_socket.getsockname()}")

            print("You > ", end="", flush=True)

        elif notified\_socket == sys.stdin:

            # Not a socket, instead it's the user writing something

            message = sys.stdin.readline().strip()

            # If not message (eg: \n) don't send it

            if message:

                # TODO: Check if message + username + data > RECV\_BUFFER

                data["message"] = message

                # print(f"Sending {data}")

                data\_send = json.dumps(data)

                data\_send = bytes(data\_send, "utf-8")

                peer: socket.socket

                for peer in sockets\_list:

                    if peer != sock\_server and peer != sys.stdin:

                        peer.send(data\_send)

                print("You > ", end="", flush=True)

        else:

            # Peer sending msg

            message = receive\_message(notified\_socket)

            if not message:

                # Peer disconnected

                delete\_last\_line()

                print(f"Peer disconnected {notified\_socket.getsockname()}")

                print("You > ", end="", flush=True)

                sockets\_list.remove(notified\_socket)

                continue

            delete\_last\_line()

            print(f"{message['username']} : {message['message']}")

            print("You > ", end="", flush=True)