MEMORIA P2

MONTERROSO BARCO, ALBERTO

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

CLIENTE-SERVIDOR

```python
# 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}")

# 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(1024)
        # 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:
    # 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)

            # 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]

            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:

                    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


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]
```

```python
PORT = int(sys.argv[2])

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:
```

```
            # 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(1024)
            # 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
```

P2P

```python
# 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}")

# 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(1024)
        # 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()

            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 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]
```

```python
PORT = int(sys.argv[2])

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)


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")

sockets_list = [sock]
client_list = {}
client_connections_list = {}


while True:
    read_sockets, _, exception_sockets = select.select(
        sockets_list, [], sockets_list)
```

```
    for notified_socket in read_sockets:

        if notified_socket == sock:
            data_received = sock.recv(1024)
            # 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)
            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("Listening...")
                    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
                    # TODO: Address already in use
                    print(f"Port: {port}")
                    server_socket.bind((ip, port))
                    server_socket.listen()
                    sockets_list.append(server_socket)
            else:
                # Message from client
                pass


# Source: https://github.com/engineer-man/youtube/blob/master/141/client.py
```