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}")
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)
```

```python
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)
```

```python
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)
```

```python
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)
```

```python
        # 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)
```

```python
# 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)
```