

Práctica 2: Programación con sockets

Autores:

Alejandro Serrano Fernández

Pedro Antonio Navas Luque

Introducción:

En esta memoria describiremos todos los programas solicitados en el Practica 2 de la asignatura para practicar con la librería socket de Python. Esta memoria está realizada por los alumnos Alejandro Serrano Fernández y Pedro Antonio Navas Luque. Cabe destacar que todas las pruebas del código están realizadas bajo Python 2.7

Programa 1: Probar Sockets

El primer programa será implementar un programa cliente/servidor que comunique dos procesos. Probar las diferencias que ocurren si el socket es TCP o UDP. ¿Qué diferencias hay? ¿Qué pasa si el cliente se inicia antes que el servidor en el caso TCP y UDP?

Como podemos observar, en el protocolo UDP no es necesario que el servidor inicie antes que el cliente para poder correr el programa. Por otra parte, no es necesario establecer una conexión entre ambos, es decir, el cliente puede enviar mensajes al servidor y éste podría no recibirlos o perder parte de la información del mensaje del cliente.

```
import socket
import time

udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

message = raw_input("Envia mensaje: ")
udp.sendto(message, ("localhost", 1207))
print("Mensaje enviado")

udp.close()
```

Figura 1: Cliente UDP

```
import socket
import time

udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp.bind(("localhost", 1207))

data, addr = udp.recvfrom(1024)
print(data)

udp.close()
```

Figura 2: Servidor UDP

```

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 2058))

s.send("Hola mundo!")

mensaje = s.recv(1024)
print(mensaje)
s.close()

```

Figura 3: Cliente TCP

```

import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('localhost', 2058))
sock.listen(1)

socket_cliente, addr = sock.accept()

mensaje = socket_cliente.recv(1024)
print(mensaje)

sock.close()
socket_cliente.close()

```

Figura 4: Servidor TCP

Sin embargo, con el protocolo TCP el servidor y el cliente establecen una conexión antes de iniciar a transmitir entre ambos, esto permite que el mensaje se envíe con total seguridad, sin perder ninguna información del mensaje enviado, pues TCP se apoya sobre la utilización de acuses de recibo para notificar si un ordenador u otro ha perdido información del mensaje. En ese caso el ordenador origen enviará de nuevo la parte corrupta del mensaje. Otra de las comparaciones con UDP en este chat simple, es el orden de ejecución de los programas, es decir, que para iniciar el chat entre ambos, es necesario que el servidor inicie primero que el cliente, de lo contrario no se podrá establecer una conexión entre ambos terminales.

Programa 2: Chat Simple

Realizar un chat simple, en principio síncrono: un usuario habla, y se espera a que el otro usuario responda. Investigar qué función de Python hay para leer por teclado. Implementar una versión con TCP y otra con UDP. ¿Qué diferencias hay entre ellas? Describir qué flujo ocurre entre los participantes.

```
import socket

udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

nombre = raw_input("Introduce tu Nombre: ")
udp.sendto(nombre, ("localhost", 2502))

nombre_server, addr = udp.recvfrom(1024)
print(nombre_server)

check = True
while check:
    message = raw_input(nombre+": ")

    if message == "exit":
        udp.sendto(message, ("localhost", 2502))
        check = False
        print("chat finalizado")
        udp.close()
    else:
        udp.sendto(message, ("localhost", 2502))

    data, addr = udp.recvfrom(1024)

    if data == "exit":
        check = False
        print("chat finalizado")
        udp.close()
    else:
        print(nombre_server+": "+data)
```

Figura 1: Programa de prueba cliente (UDP)

```
import socket

udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

nombre = raw_input("Introduce tu Nombre: ")
udp.bind(("localhost", 2502))

nombre_cliente, addr = udp.recvfrom(1024)
udp.sendto(nombre, addr)

check = True
while check:
    data, addr = udp.recvfrom(1024)

    if data == "exit":
        check = False
        print("chat finalizado")
        udp.close()
    else:
        print(nombre_cliente+": "+data)
        message = raw_input(nombre+": ")

        if message == "exit":
            check = False
            udp.sendto("exit", addr)
            udp.close()
        else:
            udp.sendto(message, addr)
```

Figura 2: Programa de prueba servidor (UDP)

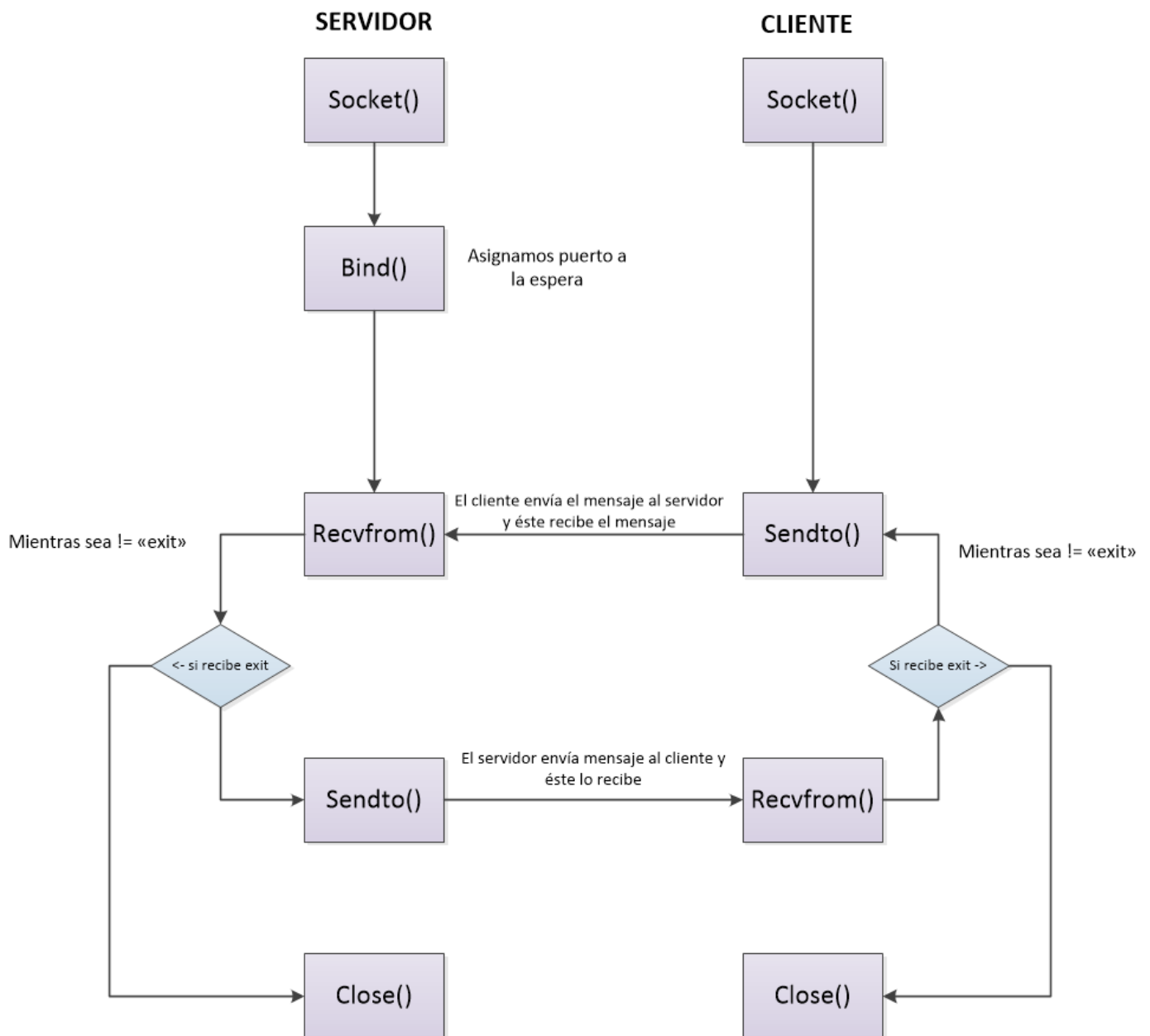


Figura 3: Flujo del chat con UDP

```

import socket
import time
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

nombre = raw_input("Introduce tu nombre: ")

s.connect(('localhost', 2085))
time.sleep(1)

server_name = s.recv(1024)
s.send(nombre)

check = True
while check:

    message = raw_input(nombre+" : ")

    if message == "exit":
        check = False
        s.send(nombre+" ha salido del chat")
        s.close()

    else:
        s.send(message)

    server_message = s.recv(1024)

    if server_message == "exit":
        check = False
        print(server_name+ " ha salido del chat")
        s.close()

    else:
        print(server_name+": "+server_message)

```

Figura 4: Cliente TCP

```

import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

nombre = raw_input("Introduce tu nombre: ")

sock.bind(('localhost', 2085))
sock.listen(1)

socket_cliente, addr = sock.accept()

socket_cliente.send(nombre)
nombre_cliente = socket_cliente.recv(1024)

print("-----"+nombre_cliente+" se ha unido-----")

check = True
while check:

    mensaje = socket_cliente.recv(1024)

    if mensaje == "exit":
        check = False
        print(nombre_cliente + " ha salido del chat")
        sock.close()
        socket_cliente.close()

    else:
        print(nombre_cliente + ": " + mensaje)

    enviar = raw_input(nombre+": ")

    if(enviar == "exit"):
        check = False
        socket_cliente.send(nombre+" ha salido del chat")
        sock.close()
        socket_cliente.close()

    else:
        socket_cliente.send(enviar)

```

Figura 5: Servidor TCP

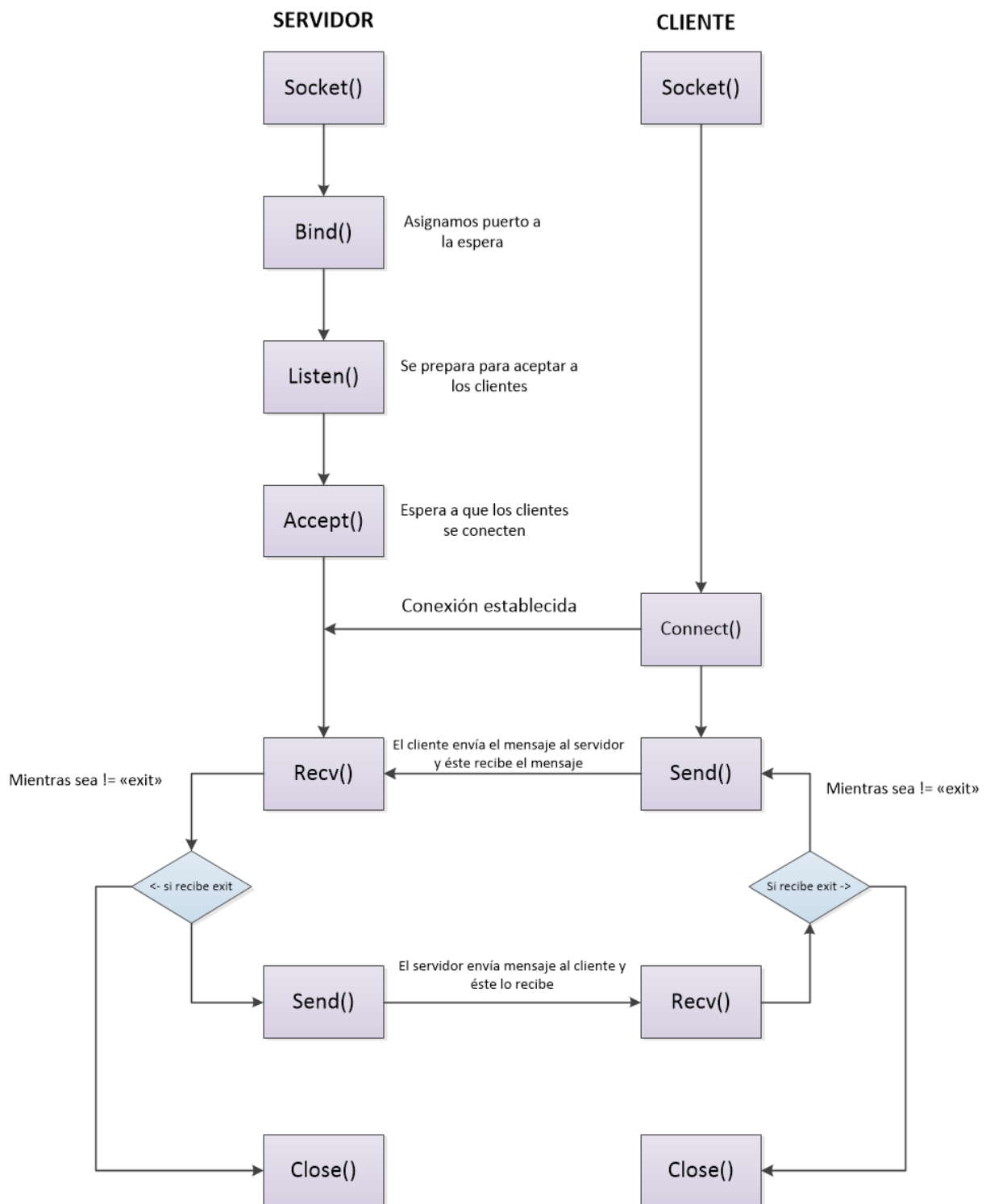


Figura 6: Flujo del chat con TCP

La diferencia entre ambos estriba en la conexión, es decir, con TCP aseguramos que el mensaje llega a su destino, esto es debido a como se comentaba en el punto anterior, TCP utiliza acuses de recibo para garantizar la entrega. Además, TCP es un protocolo orientado a conexión, pues antes de comenzar a transmitir se realiza un saludo de 3 vías con el que los equipos se sincronizan para el correcto funcionamiento de la transmisión.

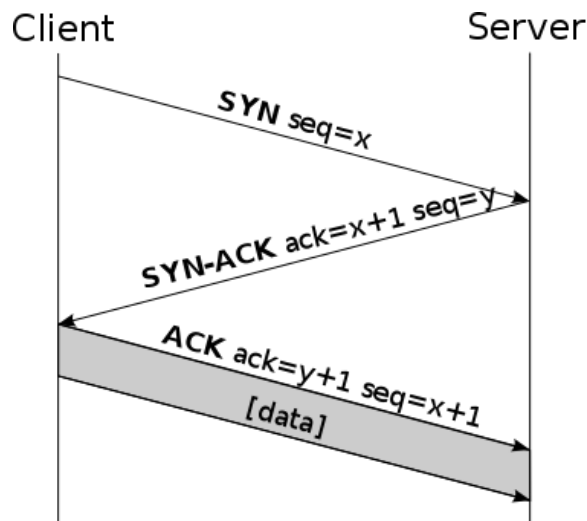


Figura 7: Saludo de 3 vías TCP

Centrándonos en el ejemplo del chat, con el protocolo TCP, aseguramos que los mensajes que envíen cada uno de los participantes sean entregados correctamente a los demás. Su desventaja es ligeramente más lento que el protocolo UDP, aunque en nuestro caso no nos afectaría mucho, pues estamos trabajando con pocos bytes.

En cambio, con el protocolo UDP, la garantía de la entrega y el control podrían estar perdidas, pues no es un protocolo orientado a conexión. Frecuentemente UDP es utilizado para aplicaciones que requieran velocidad y toleren pequeñas pérdidas de datos o aplicaciones que transmitan grandes cantidades de datos.

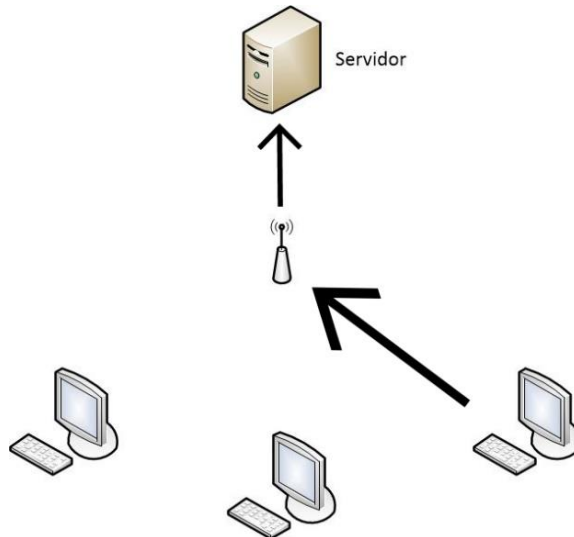
A diferencia con TCP, un servicio UDP, soporta mayor número de clientes, pues al contrario que TCP, ésta no mantiene información de estado de cada conexión (número de secuencia, ACK, ...), ni provoca retardos debido a las fases de establecimiento y finalización de conexión.

En conclusión, no hemos notado mucha diferencia entre ambos protocolos, tanto TCP como UDP el chat nos ha funcionado correctamente, sin pérdidas de datos. Aunque algo que podemos destacar entre ambos, es que en TCP es totalmente necesario que el servidor inicie antes que el cliente para su correcto funcionamiento, pues el cliente desea establecer conexión con el servidor, y si éste no está disponible, el programa finaliza sin éxito, mientras que con UDP, el cliente puede iniciar antes que el servidor aunque éste no esté disponible.

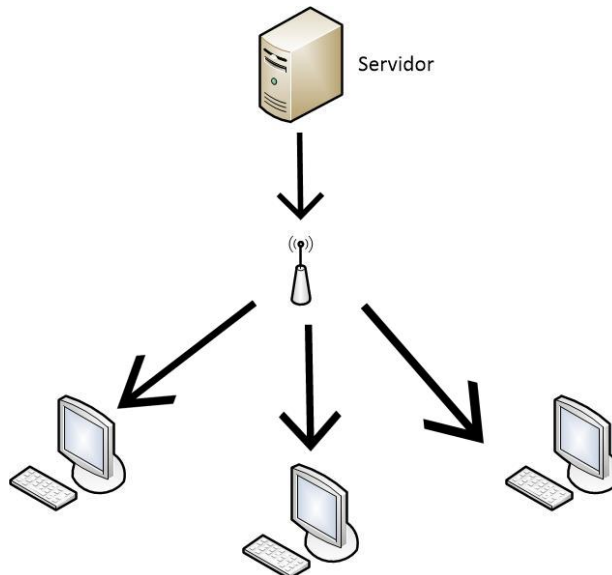
2.1 Mejora del chat simple (con TCP):

En la mejora del chat simple, hemos añadido la funcionalidad de poder añadir múltiples usuarios en el chat. Esta mejora la hemos implementado de la siguiente manera:

1. El cliente envía el mensaje que quiere hacer llegar a todos los usuarios hacia el servidor.



2. El servidor transmite el mensaje hacia todos los usuarios del chat.



2.1.1 Programa TCP servidor

```
import socket
import threading

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('localhost',1045))
sock.listen(1)

clientes = []
apodos = []
```

En primer lugar, importaríamos las librerías socket y threading, la primera para establecer conexión con los demás usuarios a través de TCP, y la segunda para permitir que varios usuarios puedan enviar y recibir mensajes a la vez.

Posteriormente asignamos un socket para nuestro servidor y asignamos el puerto a la espera para que los clientes se unan.

Las listas de clientes y apodos las declaramos para poder almacenar todos los sockets de los clientes y sus correspondientes apodos.

```
def aceptar_clientes():
    while True:
        socket_cliente, addr = sock.accept()
        socket_cliente.send("intr_apodo")
        nick = socket_cliente.recv(1024)
        apodos.append(nick)
        clientes.append(socket_cliente)
        enviar_a_todos(nick+" se ha unido al chat\n")
        print("~"+nick+" se ha unido al chat ~")
        socket_cliente.send("Te has unido al chat")

        thread = threading.Thread(target=recibir, args=(socket_cliente,))
        thread.start()

def recibir(cliente):
    check = True
    while check:
        try:
            data = cliente.recv(1024)

            if data == "salir":
                indice = clientes.index(cliente)
                apodo_indice = apodos[indice]
                eliminar_cliente(cliente)
                enviar_a_todos(apodo_indice + " ha salido del chat")
                print("~"+apodo_indice + " ha salido del chat~")
                cliente.close()
                check = False
            else:
                enviar_a_todos(data)

        except:
            cliente.close()
            check = False
```

La función `aceptar_clientes()` se encarga de aceptar nuevas conexiones constantemente. Dentro de ella el servidor envía al cliente que se conecta una solicitud para que introduzca su apodo, el cual almacenaremos en nuestra lista `apodos[]`. Finalmente iniciamos un hilo para iniciar la función `recibir(cliente)`, encargada de recibir los mensajes de los clientes y enviarlo a todos los usuarios de la sala. Hay que destacar que si ésta recibe un mensaje de salida de algunos de los usuarios, el servidor lo elimina de la lista de clientes y `apodos`, cierra su conexión e indica a todos los usuarios que ha abandonado el chat.

```
def enviar_a_todos(mensaje):
    for i in clientes:
        i.send(mensaje)

def eliminar_cliente(cliente):
    indice = clientes.index(cliente)
    apodo_indice = apodos[indice]
    apodos.remove(apodo_indice)
    clientes.remove(cliente)
    cliente.close()

aceptar_clientes()
```

La función `enviar_a_todos()` es la encargada de enviar los mensajes que envían los usuarios a los demás participantes de la sala y la función `eliminar_clientes()` se encarga de eliminar los clientes de la lista de clientes y posteriormente se cierra su conexión.

Finalmente iniciamos la función `acepta_clientes()` para comenzar la ejecución del chat.

2.1.2 Programa TCP cliente

```
import socket
import threading
import time
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

nombre = raw_input("Introduce tu nombre: ")

s.connect(('localhost', 1046))
time.sleep(1)
```

En primer lugar importamos las librerías `socket`, `threading` y `time`. La primera de éstas la utilizaremos para establecer conexión con el servidor para enviar y recibir del servidor. La segunda la utilizaremos para permitir que podamos enviar y recibir mensajes de múltiples usuarios a la misma vez, y la tercera la utilizaremos únicamente para pausar el sistema antes de comenzar a

recibir datos del servidor (para evitar algunos problemas que recibíamos cuando iniciábamos el cliente).

```
def recibir():
    check = True
    while check:
        try:
            data = s.recv(1024)
            if data == "intr_apodo":
                s.send(nombre)
            else:
                print(data)
        except:
            print("IMPORTANTE: NO estas en el chat")
            check = False
            s.close()
```

La función recibir(), como su nombre indica, se encargará de recibir los datos que nos envíe el servidor. Si acabamos de entrar, el servidor nos enviará un mensaje (en nuestro caso “intr._apodo”) que indicará al programa que el servidor está a la espera de recibir nuestro apodo para poder mostrarlo en chat cada vez que escribamos un mensaje.

```
def mostrar():
    check = True
    while check:
        data = raw_input()
        if data == "salir":
            s.send(data)
            print("~ Has salido del chat ~")
            s.close()
            check = False
        else:
            s.send(nombre+": "+data)
```

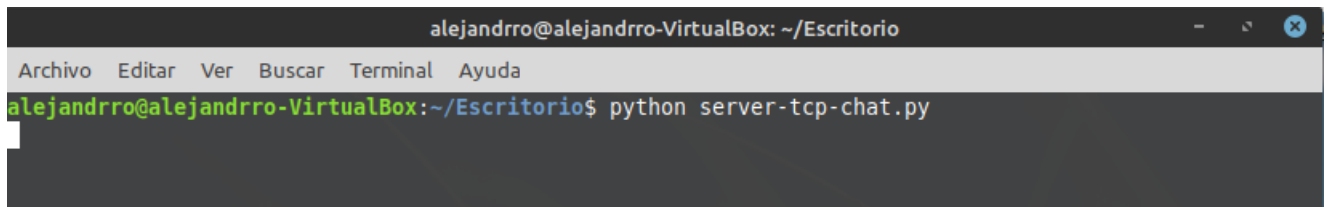
La función mostrar, se encargará de mostrar a los demás usuarios los mensajes que les enviamos. En el caso de que escribamos “salir”, el servidor cerrará nuestra sesión y nos eliminará de su lista de clientes para que no podamos enviar más mensajes por el chat.

```
recibir_thread = threading.Thread(target=recibir)
escribir_thread = threading.Thread(target=mostrar)
recibir_thread.start()
escribir_thread.start()
```

Finalmente iniciamos dos hilos, que se encargarán de ejecutar concurrentemente las dos funciones antes descrita.

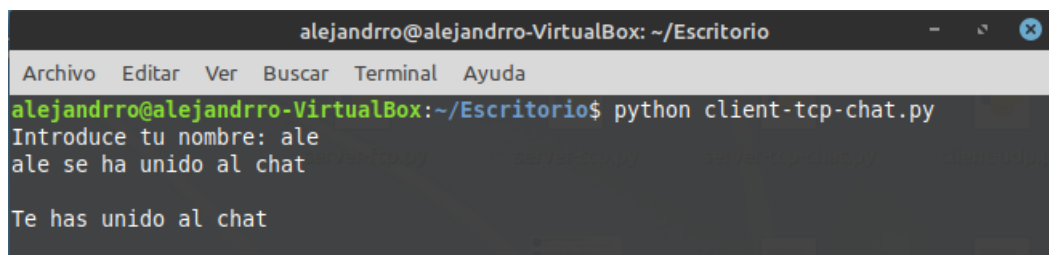
2.3. Ejecución del programa

Para poder iniciar el chat, es necesario que ejecutemos primeramente el servidor en nuestra terminal:

A screenshot of a terminal window titled 'alejandrro@alejandrro-VirtualBox: ~/Escritorio'. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The command 'python server-tcp-chat.py' has been entered and executed, resulting in a blank terminal below the command line.

```
alejandrro@alejandrro-VirtualBox: ~/Escritorio
Archivo  Editar  Ver    Buscar  Terminal  Ayuda
alejandrro@alejandrro-VirtualBox:~/Escritorio$ python server-tcp-chat.py
```

Una vez iniciamos el servidor, procedemos a iniciar todos los clientes que queramos incluir en el chat.

A screenshot of a terminal window titled 'alejandrro@alejandrro-VirtualBox: ~/Escritorio'. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The command 'python client-tcp-chat.py' has been entered and executed. The output shows the user entering the name 'ale', followed by the message 'ale se ha unido al chat', and then 'Te has unido al chat'.

```
alejandrro@alejandrro-VirtualBox: ~/Escritorio
Archivo  Editar  Ver    Buscar  Terminal  Ayuda
alejandrro@alejandrro-VirtualBox:~/Escritorio$ python client-tcp-chat.py
Introduce tu nombre: ale
ale se ha unido al chat

Te has unido al chat
```

Una vez ejecutados, introducimos el nombre de cada uno y comenzamos a enviar tantos mensajes como queramos. En la siguiente imagen se muestra una prueba del chat:

The image displays four terminal windows arranged in a 2x2 grid, all running on a system named 'alejandrro@alejandrro-VirtualBox: ~/Escritorio'. Each window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The top-left window shows the execution of 'python server-tcp-chat.py', which outputs status messages for users joining and leaving the chat. The top-right window shows 'python client-tcp-chat.py' being run by a user named 'alejandrro', with prompts for a name and subsequent chat messages. The bottom-left window continues the chat log with more messages. The bottom-right window shows the client program ending with an 'IMPORTANTE: NO estas en el chat' message.

```
alejandrro@alejandrro-VirtualBox: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
alejandrro@alejandrro-VirtualBox:~/Escritorio$ python server-tcp-chat.py
~ ale se ha unido al chat ~
~ pedro se ha unido al chat ~
~ alberto se ha unido al chat ~
~ pedro ha salido del chat~
[]

alejandrro@alejandrro-VirtualBox: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
alejandrro@alejandrro-VirtualBox:~/Escritorio$ python client-tcp-chat.py
Introduce tu nombre: ale
ale se ha unido al chat

Te has unido al chat
pedro se ha unido al chat

alberto se ha unido al chat

hola que tal?
ale: hola que tal?
alberto: hola, yo bien y ustedes?
pedro: igualmente, estoy realizando la practica 2 de sistemas distribuidos
pedro ha salido del chat
[]

alejandrro@alejandrro-VirtualBox: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
alejandrro@alejandrro-VirtualBox:~/Escritorio$ python client-tcp-chat.py
Introduce tu nombre: alberto
alberto se ha unido al chat
Te has unido al chat
ale: hola que tal?
hola, yo bien y ustedes?
alberto: hola, yo bien y ustedes?
pedro: igualmente, estoy realizando la practica 2 de sistemas distribuidos
pedro ha salido del chat
[]

alejandrro@alejandrro-VirtualBox: ~/Escritorio
Archivo Editar Ver Buscar Terminal Ayuda
alejandrro@alejandrro-VirtualBox:~/Escritorio$ python client-tcp-chat.py
Introduce tu nombre: pedro
pedro se ha unido al chat
Te has unido al chat
alberto se ha unido al chat

ale: hola que tal?
alberto: hola, yo bien y ustedes?
igualmente, estoy realizando la practica 2 de sistemas distribuidos
pedro: igualmente, estoy realizando la practica 2 de sistemas distribuidos
salir
~ Has salido del chat ~

IMPORTANTE: NO estas en el chat
alejandrro@alejandrro-VirtualBox:~/Escritorio$ []
```

La mejora con UDP no la hemos podido realizar correctamente, pues recibíamos muchos errores al ejecutarla, aún así está realizado e incluido en el .zip de la práctica

Programa 3: FTP simple

Escribir un programa cliente/servidor en el que el servidor recibe por socket el nombre de un fichero y se lo envía al cliente.

3.1 Funcionamiento del programa

En este programa primeramente conectaremos el cliente/servidor mediante sockets TCP, una vez conectados el cliente este recibirá una lista de los ficheros disponibles del servidor, y ya podrá realizar peticiones mediante comandos, estos comandos son:

- ➡ “list” :solicitará la lista de ficheros disponibles del servidor. Obtenemos esta lista mediante la librería os que nos permite acceso a funciones de Linux tal como “ls”.
- ➡ “exit” :finalizará la sesión.
- ➡ “subir”: pedirá al cliente que archivo txt quiere subir, al introducirlo el servidor lo recibirá y actualizará la lista de archivos disponibles.

Y si introducimos cualquier otra cosa, lo tomará como un nombre de archivo y lo pedirá al servidor, el cual comprobará la lista de archivos disponibles. Si se encuentra en la lista, le enviará el contenido al cliente, el cual en caso de que no tenga ya dicho archivo creará un archivo con el mismo nombre con la función “touch” y copiará el contenido en él, en caso contrario simplemente modificará el contenido del archivo. Si no se encuentra en la lista devolverá “el archivo solicitado no existe”. En el código podemos ver unos s.send(“recibido”) por parte del servidor y confirmaciones=s.recv(1024) por parte del cliente, esto lo hacemos con el propósito de seguir la secuencialidad de cliente/servidor (recibe-envía-recibe...).

3.2 Programa del servidor FTP

```
import socket
import os

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Estamos a la espera hasta que se establezca conexion
sock.bind(('localhost', 1055))
sock.listen(1)

socket_cliente, addr = sock.accept()
#Recibimos el nombre del cliente
nombre_cliente = socket_cliente.recv(1024)

print("-----"+nombre_cliente+" se ha conectado-----")

#Guardamos en un fichero el listado de los ficheros del directorio.
#Si el fichero no existe crea uno nuevo, o lo sobrescribe.
os.system("ls > list.txt")
file = open("list.txt", 'rb')
data_file = file.read(1024)

#Enviamos al cliente el listado de los ficheros
socket_cliente.send(data_file)
```

Inicialmente importamos las librerías socket y os, la primera para conectarnos al servidor y la segunda para realizar comandos propios de Linux.

Posteriormente abrimos asociamos el puerto y esperamos a que se conecte el cliente. Una vez conectado, recibimos su nombre y le enviamos el listado de archivos del directorio en el que se encuentra el programa.


```

check = True
while check != False:

    mensaje = socket_cliente.recv(1024)
    print("El usuario "+nombre_cliente+" ha realizado el siguiente comando: "+mensaje)

    if mensaje == "list":
        os.system("ls > list.txt")
        file = open("list.txt", 'rb')
        data_file = file.read(1024)
        file.close()
        socket_cliente.send(data_file)

    elif mensaje == "exit":
        socket_cliente.send("Sesion finalizada")
        check = False

    elif mensaje == "subir":
        socket_cliente.send("recibido")
        mensaje=socket_cliente.recv(1024)
        socket_cliente.send("recibido")
        os.system("touch " +mensaje)
        os.system("ls > list.txt")
        cliente_message = socket_cliente.recv(1024)
        file = open(mensaje, 'wb')
        file.write(cliente_message)
        file.close()

    else:
        flist=open("list.txt", 'r')
        comprueba=False
        fichero = flist.readline()
        while fichero != "":
            if (mensaje+'\n')==fichero:
                comprueba = True
                fichero=flist.readline()
        flist.close()
        if comprueba==False:
            socket_cliente.send("error")
        else:
            file = open(mensaje, 'rb')
            data_file = file.read(1024)
            file.close()
            socket_cliente.send(data_file)

```

Una vez enviado el listado de archivos del directorio, el servidor está a la espera de los comandos que le envíen los clientes para proporcionarle lo solicitado por él. El servidor estará en funcionamiento hasta que el cliente no finalice con el comando “salir”.

3.3 Programa del Cliente FTP

```

import socket
import os

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

nombre = raw_input("Introduce tu nombre: ")

s.connect(('localhost', 1055))

#Enviamos nuestro nombre al servidor
s.send(nombre)

#Recibimos un listado de ficheros del directorio del servidor
server_message = s.recv(1024)
print(server_message)

```

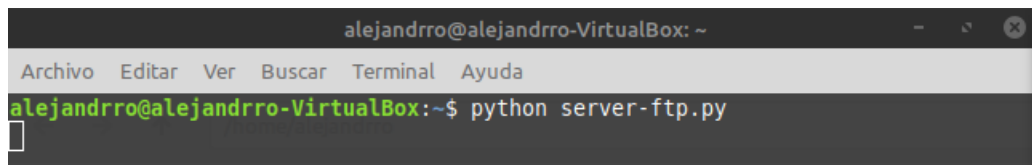
Importamos las dos mismas librerías antes comentadas para el correcto funcionamiento del programa.

Inicialmente el cliente introduce su nombre y se conecta al servidor FTP al que le enviará dicho nombre.

Una vez conectado el cliente recibe el listado de los ficheros del directorio en el se encuentra el servidor. Posteriormente el cliente podrá realizar alguno de los comandos anteriormente descritos en la sección **3.1** de la memoria

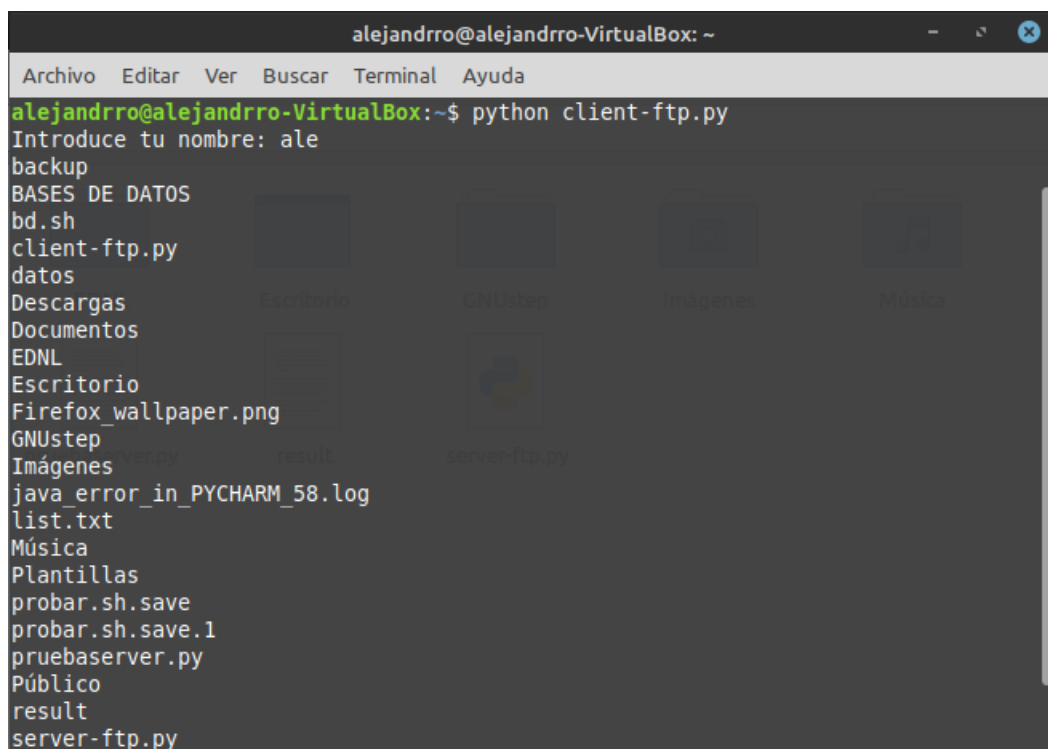
3.3 Ejecución del programa

Inicialmente ejecutamos el programa FTP del servidor en la terminal:



```
alejandrro@alejandrro-VirtualBox: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
alejandrro@alejandrro-VirtualBox:~$ python server-ftp.py  
█
```

Posteriormente ejecutamos el programa FTP del cliente en otra terminal e introducimos el nombre del cliente:



```
alejandrro@alejandrro-VirtualBox: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
alejandrro@alejandrro-VirtualBox:~$ python client-ftp.py  
Introduce tu nombre: ale  
backup  
BASES DE DATOS  
bd.sh  
client-ftp.py  
datos  
Descargas  
Documentos  
EDNL  
Escritorio  
Firefox_wallpaper.png  
GNUstep  
Imágenes  
java_error_in_PYCHARM_58.log  
list.txt  
Música  
Plantillas  
probar.sh.save  
probar.sh.save.1  
pruebaserver.py  
Público  
result  
server-ftp.py
```

Una vez ejecutado recibimos todos los ficheros del directorio del servidor y nos dará la opción de escribir alguno de los comandos ofrecidos anteriormente:

```
Público  
result  
server-ftp.py  
Videos  
Comando: 
```