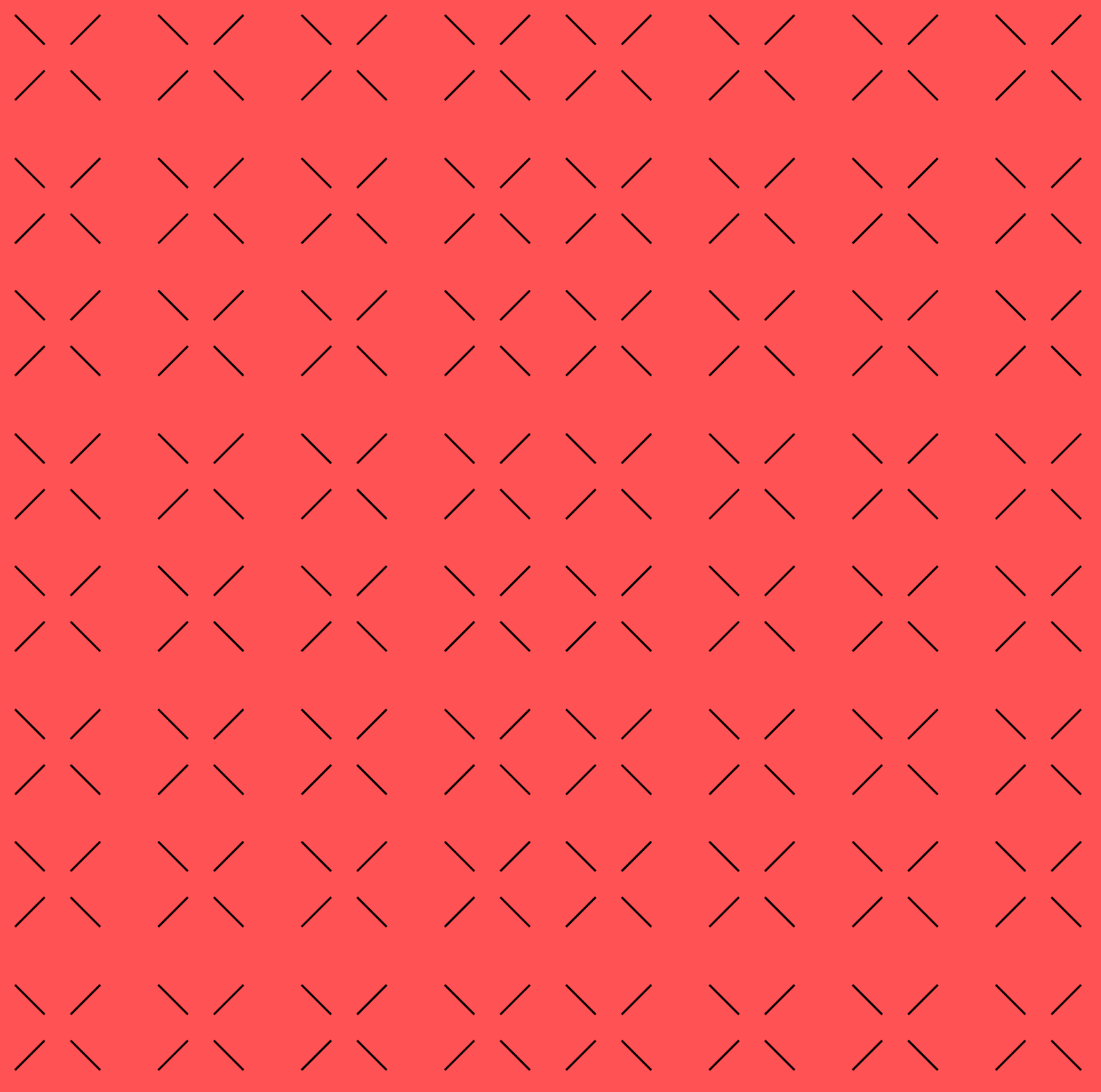


Unidad 2.2

Programación de comunicaciones en red



Índice

Sumario

1. Introducción a los Sockets Cliente-Servidor en Python.....	4
1.1. Ejemplo 1: Pedir la hora a un servidor NTP.....	5
1.2. Ejemplo 2: Solicitar una página web a un servidor.....	7
2. Sockets y DNS.....	9
2.1. Ejemplo de consultas DNS con socket.....	9
3. Socket cliente-servidor orientado a conexión: TCP.....	10
3.1. Ejercicio Cliente – Servidor TCP.....	12
3.2. Problema a resolver con un ejercicio entregable.....	16

Licencia



Reconocimiento – NoComercial – CompartirIgual (by-nc-sa):

No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

1. Introducción a los Sockets Cliente-Servidor en Python

Los sockets son una interfaz de programación que permite la comunicación entre dos procesos a través de una red. En este contexto, el proceso que proporciona la información o los servicios es el servidor, y el proceso que solicita y recibe estos servicios es el cliente.

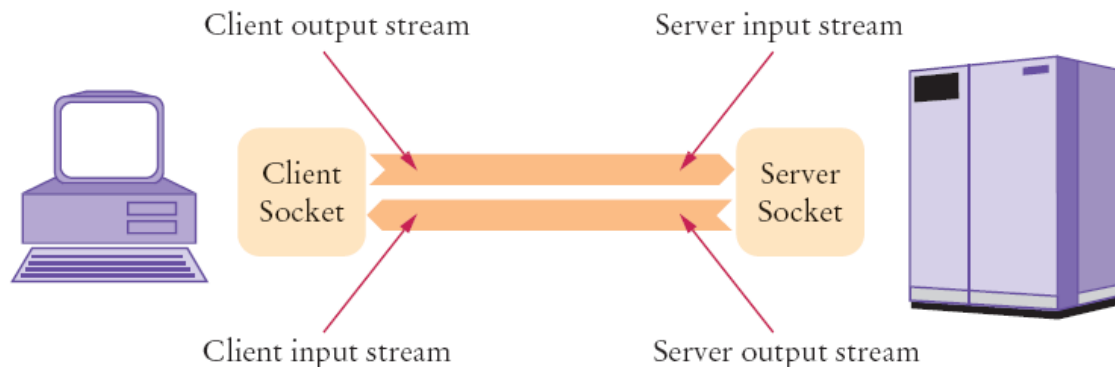


Figure 5 Client and Server Sockets

Los datos que se envían los procesos mediante los sockets no son caracteres ni strings. En Python los mensajes que se envían el cliente y es servidor son **cadenas de bytes**.

Haz esta prueba:

```
b = b"Mensaje que se convierte a binario"
print(b) # imprime la b (no imprime el cuerpo real del mensaje)
print(str(b)) # imprime la b (no imprime el cuerpo real del mensaje)
print(b.decode()) # ahora si imprime le mensaje,
                  # porque lo hemos decodificado de byte a string
```

Verás que sólo cuando decodificas el mensaje en binario se imprime correctamente.

Por ello:

- Cada vez que recibimos información de un servidor, hemos de decodificar o formatear la respuesta antes de imprimirla por pantalla.
- Cada vez que enviamos información a un servidor, la tenemos que codificar o adaptar al formato que espera el servidor.

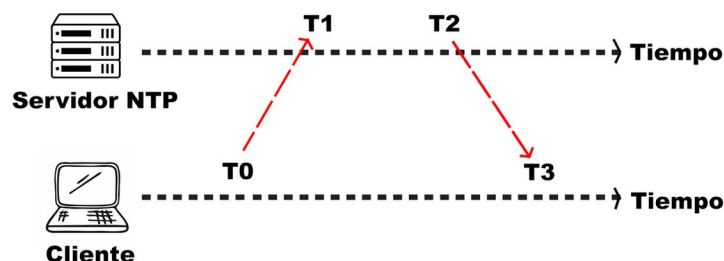
1.1. Ejemplo 1: Pedir la hora a un servidor NTP

Un servidor NTP (Network Time Protocol) es un sistema que proporciona servicios de sincronización de tiempo a través de una red. Su función principal es mantener una referencia precisa del tiempo y distribuir esta información a los clientes que la solicitan. Estos servidores están conectados a fuentes de tiempo altamente precisas, como relojes atómicos o satélites GPS, para asegurar la coherencia y la exactitud en la marcación temporal.

Los clientes NTP pueden sincronizarse con estos servidores para ajustar sus relojes internos, garantizando así que múltiples sistemas en una red tengan una noción coherente del tiempo. La sincronización precisa del tiempo es esencial en una variedad de aplicaciones, desde transacciones financieras hasta la coordinación de eventos distribuidos en redes informáticas.

NTP utiliza **UDP** como su capa de transporte, usando el **puerto 123**. Está diseñado para resistir los efectos de la latencia variable.

El nodo del cliente envía un mensaje de solicitud en el momento t_1 al nodo del servidor y al servidor envía el valor de tiempo actual en el mensaje de respuesta en el momento t_3 .



Autor: RaulSantosRecio - <https://commons.wikimedia.org/w/index.php?curid=118061937>

Comportamiento de cada parte:

- **Servidor:**
 - Escucha.
 - Recibe una petición, como es UDP **se guarda el remitente**.
 - Contesta con la hora **al remitente**.
 - Cierra la conexión.
- **Cliente:**
 - Se conecta al servidor.
 - Espera que le responda con la hora, recibe el remitente, pero no lo usaremos.
 - Cierra la conexión.

```
import socket
import struct
import time

def get_ntp_time(socket,ntp_server,ntp_port):
    """ Función que dado un socket comunica con servidor NTP para recibir la hora actual."""

    # Preparar el mensaje NTP (protocolo SNTP) que se enviará al servidor
    ntp_request = b'\x1b' + 47 * b'\0'

    try:
        # El cliente envía un mensaje de solicitud al servidor NTP
        socket.sendto(ntp_request, (ntp_server, ntp_port))

        # El cliente recibe la respuesta del servidor
        ntp_response, server_address = socket.recvfrom(1024)

    except socket.error as e:
        print(f"Error al conectar al servidor NTP: {e}")
        return None
    finally:
        # El cliente cierra el socket (el servidor probablemente ya
        # ha cerrado su parte de la comunicación)
        client_socket.close()

    # Gestionamos la respuesta del servidor
    unpacked_response = struct.unpack('!12I', ntp_response)
    ntp_seconds = unpacked_response[10] - 2208988800 # Epoch time (1900-1970)

    # Convertir a formato imprimible
    tiempo_servidor = time.strftime('%Y-%m-%d %H:%M:%S', time.gmtime(ntp_seconds))

    return tiempo_servidor

if __name__ == "__main__":
    # Obtener y mostrar la hora del servidor NTP

    # Servidor NTP de google
    ntp_server = 'time.google.com'
    ntp_port = 123

    # Crear el socket del cliente
    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    except socket.error as e:
        print(f"Error al crear el socket UDP: {e}")

    ntp_hora = get_ntp_time(client_socket, ntp_server, ntp_port)

    if ntp_hora:
        print(f"Hora del servidor NTP: {ntp_hora}")
    else:
        print("No se pudo obtener la hora del servidor NTP.")
```

1.2. Ejemplo 2: Solicitar una página web a un servidor

Vamos a solicitar una página web a un servidor llamado: example.com

Como es un servidor web y usaremos una conexión no segura, nos conectaremos al puerto 80 del servidor usando el protocolo TCP.

Comportamiento de cada parte:

- **Servidor:**
 - Escucha.
 - Recibe una petición. Como es una conexión TCP, no hace falta guardar el remitente.
 - El servidor espera la solicitud del cliente para saber qué página quiere ver.
 - El cliente le envía que quiere ver y el servidor responde con la página que han solicitado.
 - Cierra la conexión. (En este caso)
- **Cliente:**
 - Se conecta al servidor.
 - Le envía la solicitud de la página que quiere ver.
 - Recibe la página web que habíamos solicitado.
 - Imprime la página web por pantalla, en un caso real nuestro navegador nos mostraría la página html.
 - Cierra la conexión.

Como hemos comentando en el punto 1 tanto los mensajes que enviamos al servidor como los que recibimos, se han de formatear y a veces decodificar (porque la información está en binario).

El formato para pedir la página principal del sitio es:

```
peticion = b'GET / HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n'
```

Funciones para enviar información:

- **send:** se utiliza para enviar datos a través de un socket. Se indican cuantos bytes queremos enviar y la función nos devuelve el número real de bytes enviados.
bytes_enviados = send() devuelve cuantos bytes ha logrado enviar, pero puede que no sea todo el mensaje o petición enviada, sino sólo una parte. En nuestro caso es una petición de tamaño muy reducido y por ello da lo mismo s.send(peticion) que s.sendall(peticion)
- **sendall:** es similar a send, pero envía todo el bloque de información que se quiere enviar.

Función para recibir información:

- **recv:** devuelve el mensaje recibido en bytes. Se le indica cuantos bytes queremos recibir. En estos ejercicios pondremos 1024.

El código que os muestro, le falta una cosa. No se baja toda la página web, sólo una parte:

```
import socket

def pedir_página_web(socket,web_path):
    """Obtiene y muestra por pantalla una página web del path solicitado."""

    # Formateamos el mensaje de solicitud HTTP GET
    petición = f'GET {web_path} HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n'

    try:
        # Enviamos la petición al servidor
        socket.sendall(petición.encode())
        # Recibimos y mostramos la respuesta del servidor
        mensaje = socket.recv(1024)
        # Mostramos la respuesta del servidor
        print(mensaje.decode())
    except socket.error as exc:
        print ("Excepción de socket: %s" % exc)
        raise
    finally:
        socket.close()

if __name__ == "__main__":
    # Configuración del servidor web
    web_host = "example.com"
    web_port = 80
    web_path = "/" #solicitamos la página principal, la raíz

    try:
        socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error as exc:
        print ("Error al crear el socket: %s" % exc)
        raise

    try:
        socket_cliente.connect((web_host,web_port))
    except socket.error as exc:
        print ("Error al conectar el socket: %s" % exc)
        raise
    print('Conectado con éxito')

    pedir_página_web(socket_cliente,web_path)
```

Lo trabajamos en el ejercicio 1, no evaluable.

2. Sockets y DNS

Hemos visto que para resolver un nombre el sistema consulta primer el fichero host y si no está allí, consulta el servidor DNS para obtener la IP del sitio.

A continuación hay dos ejemplos de código:

- Nos conectaremos con un socket al servidor DNS para obtener la IP de un sitio web.
- Nos conectaremos con un socket al servidor DNS para dada una IP, que nos diga a qué nombre pertenece.

2.1. Ejemplo de consultas DNS con socket

Consultar un nombre de dominio al DNS y nos devuelve su IP:

```
import socket

try:
    host = 'www.amazon.es'
    print ("IP de %s: %s" %(host, socket.gethostbyname(host)))
except socket.error as msg:
    print ("%s: %s" %(host, msg))
```

➡ IP de www.amazon.es: 18.172.228.120

Consultar una IP al DNS y nos devuelve el nombre de dominio:

```
import socket

ip = "213.201.83.138"
try:
    dominio = socket.gethostbyaddr(ip)[0]
    print ("La IP %s tiene una entrada DNS: %s" %(ip, dominio))
except socket.error as msg:
    print ("%s: %s" %(ip, msg))
```

➡ La IP 213.201.83.138 tiene una entrada DNS: www.valencia.es

Trabajaremos estas llamadas en el ejercicio 2, no evaluable.

3. Socket cliente-servidor orientado a conexión: TCP

El socket permite conectar máquinas diferentes, para intercambiar datos.

En este caso el servidor estará a la escucha de los clientes que se quieran conectar a él. Tendrá un socket escuchando en un puerto a cualquier solicitud cliente.

Cuando se acepta un cliente se le asigna un nuevo socket “exclusivo” con el que poder comunicarse cliente-servidor (misma ip, pero diferente puerto).

Los clientes enviaran mensajes al servidor y una vez finalicen, cerrarán la conexión

Ejemplo de código de un Servidor TCP:

```
import socket
import sys

if __name__ == '__main__':
    #Decidimos la IP y el puerto del servidor
    HOST = '127.0.0.1' # La IP del servidor es la loca de la máquina
    PORT = 5000 # El puerto tiene que ser superior a 1024, por debajo estan reservados

    try:
        socket_escucha = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print('Socket servidor creado')
    except socket.error:
        print('Fallo en la creación del socket servidor')
        sys.exit()

    try:
        socket_escucha.bind((HOST, PORT)) # Definimos el punto de enlace del servidor.
        # El servidor está preparado en la IP 127.0.0.1 y puerto 5000
    except socket.error as e:
        print('Error socket: %s' % e)
        sys.exit()

    socket_escucha.listen(5) # El servidor puede escuchar hasta 5 clientes.

    while True:
        socket_atiende, addr_cliente = socket_escucha.accept() #El Servidor queda bloqueado en esta línea
        # esperando a que un cliente se conecte a su IP y puerto
        # Si un cliente se conecta guardamos en socket_atiende el nuevo socket
        # y en addr_cliente la información del cliente (IP y puerto del cliente)

        fin_mensaje = b''

        with socket_atiende:
            #print(f"Conexión exitosa con el cliente. IP ({addr[0]}) Puerto ({addr[1]})")
            print(f"Conexión exitosa con el cliente. {addr_cliente}")
            cerrar = False
            while not cerrar:
                data = socket_atiende.recv(1024) #El servidor queda bloqueado esperando el mensaje que le va a enviar el cliente
                if data==fin_mensaje:
                    cerrar = True
                else:
                    print(f'El cliente {addr_cliente} nos ha escrito: {data.decode()}') #Mensaje recibido, lo imprimimos
                    socket_atiende.sendall(b"mensaje recibido")
            print("Fin de conversación con: ",addr_cliente)
            socket_atiende.close()
```

Ejemplo de código de un Cliente TCP:

```
import socket
import sys

HOST = '127.0.0.1'
PORT = 5000

def programa_cliente():
    try:
        socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        print('Socket cliente creado')
    except socket.error:
        print('Fallo en la creación del socket cliente')
        sys.exit()

    socket_cliente.connect((HOST, PORT))

    mensaje = 'Conectado con el servidor' # El programa cliente nos pide que escribamos algo

    with socket_cliente:
        while mensaje != 'bye':
            socket_cliente.sendall(mensaje.encode()) # Codificamos el mensaje a bytes
                                                    # y le indicamos que lo envíe todo
            data = socket_cliente.recv(1024) #línea bloqueante, esperamos que el servidor nos conteste
            print('Recibido del servidor:' + data.decode())
            mensaje = input('Escribe tu mensaje (Para finalizar escribe: bye)--> ')

if __name__ == '__main__':
    programa_cliente()
```

3.1. Ejercicio Cliente – Servidor TCP

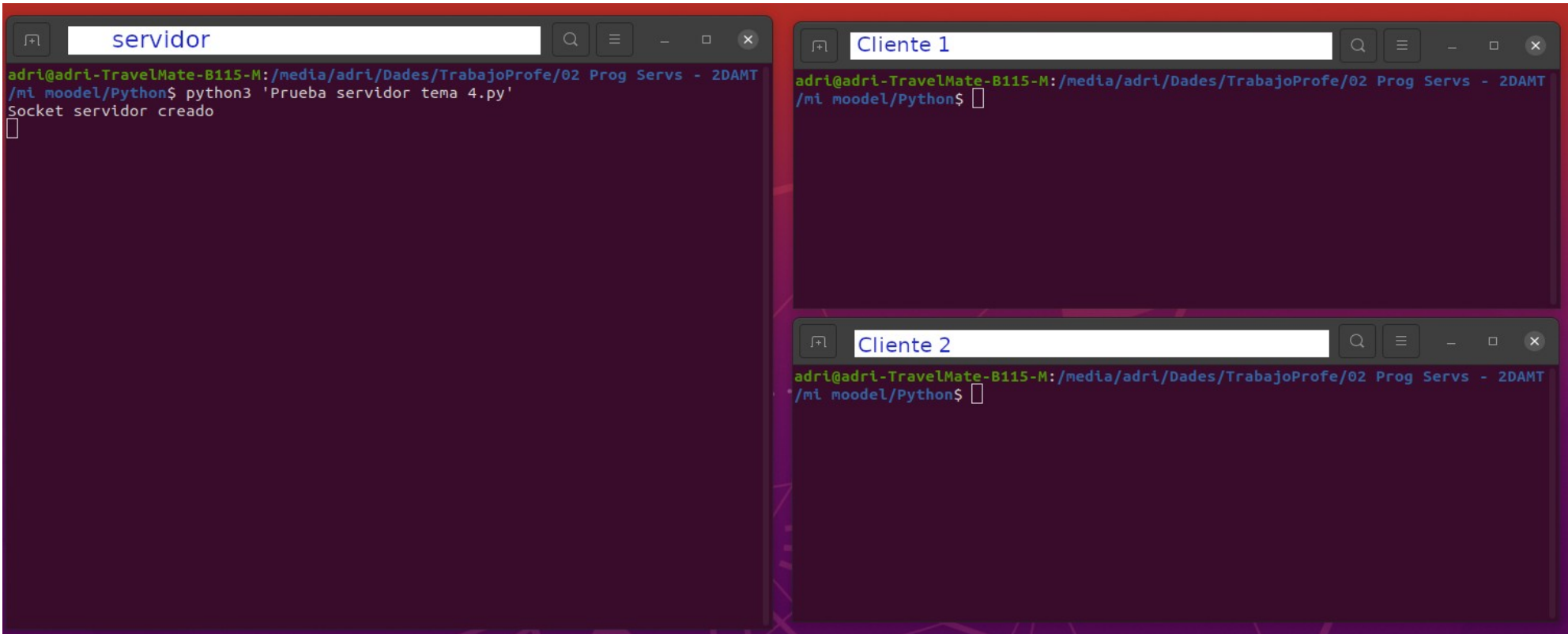
Para probar este ejercicio tenemos que:

1. Ejecutar el programa servidor para que ponga nuestro programa a la escucha del puerto que hayamos definido. En nuestro caso será siempre un puerto privado o dinámico (un número de puerto mayor que 5000) y como no servimos fuera usamos la ip interna del ordenador de loopback 127.0.0.1
2. Ejecutamos el cliente: Tiene que saber la ip del servidor (en nuestro caso es la: 127.0.0.1) y el puerto en el que está escuchando el servidor (en este ejemplo está en el puerto 5000).
3. El servidor recibe la solicitud del cliente y acepta la conexión.
4. Al aceptar la conexión, crea un nuevo extremo de socket con otro puerto privado aleatorio y establece la conexión con el cliente. En el ejemplo que os pondré a continuación el servidor habla con el cliente desde el socket con ip:127.0.0.1 y puerto: 44104.
5. El cliente al recibir la 3 way hand shake de la ip:127.0.0.1 con puerto: 44104, fija la conexión con este puerto.
6. Chatean un rato el cliente y el servidor.
7. El cliente escribe bye para indicar al servidor que cierra la comunicación.
8. El cliente cierra su lado del socket
9. El servidor recibe el bye y cierra su lado del socket

En los ejercicios os plantearé un problema que tendréis que resolver.

Voy a conectar 2 clientes. Hay que abrir en total 3 terminales, uno para el servidor y un terminal para cada cliente:

- Arranco el servidor en un terminal.
- Preparo 2 terminales para luego ejecutar el programa cliente 2 veces



The image shows three terminal windows side-by-side. The left window, titled 'servidor', shows the command `python3 'Prueba servidor tema 4.py'` being executed, resulting in the output 'Socket servidor creado'. The right window has two sub-panels. The top sub-panel, titled 'Cliente 1', shows the command `/mi moodel/Python$` being entered. The bottom sub-panel, titled 'Cliente 2', also shows the command `/mi moodel/Python$` being entered. All three windows have a dark purple background and a light blue prompt.

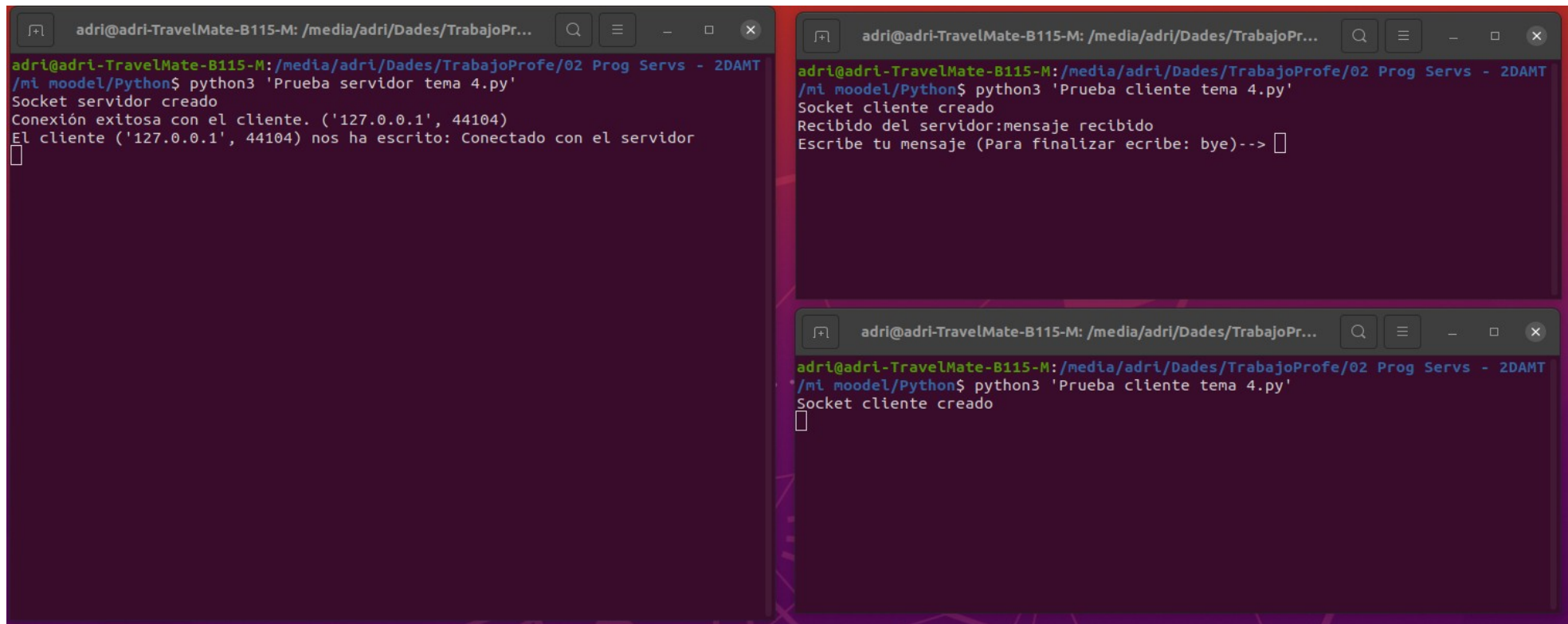
```
servidor
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba servidor tema 4.py'
Socket servidor creado

Cliente 1
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$

Cliente 2
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$
```

En este punto:

- Se ha ejecutado el programa cliente 1
- El servidor lo ha aceptado y lo atiende en el puerto 44104
- Al ejecutar el cliente 2, vemos que el servidor no lo atiende, tiene que esperar.



```
adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba servidor tema 4.py'
Socket servidor creado
Conexión exitosa con el cliente. ('127.0.0.1', 44104)
El cliente ('127.0.0.1', 44104) nos ha escrito: Conectado con el servidor
█

adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba cliente tema 4.py'
Socket cliente creado
Recibido del servidor:mensaje recibido
Escribe tu mensaje (Para finalizar escribe: bye)--> █

adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba cliente tema 4.py'
Socket cliente creado
█
```


Siguiente paso:

- El cliente 1 tras hablar con el servidor, acaba la conversación con bye.
- El servidor cierra la conexión con cliente 1.
- El servidor mira si hay alguien esperando en el puerto 5000.
- Está cliente 2 esperando.
- El servidor acepta al cliente 2 y le ofrece una comunicación con el y se comunica con el con el puerto: 41612
- El cliente hace 3 way hand shake con el socket destino (IP: 127.0.0.1 y puerto: 41612)
- El cliente y el servidor empiezan a intercambiar mensajes.

```
adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...  
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT  
/mi moodel/Python$ python3 'Prueba servidor tema 4.py'  
Socket servidor creado  
Conexión exitosa con el cliente. ('127.0.0.1', 50022)  
El cliente ('127.0.0.1', 50022) nos ha escrito: Conectado con el servidor  
El cliente ('127.0.0.1', 50022) nos ha escrito: Hola  
El cliente ('127.0.0.1', 50022) nos ha escrito: Soy cliente 1  
El cliente ('127.0.0.1', 50022) nos ha escrito: Me voy  
Conexión exitosa con el cliente. ('127.0.0.1', 41612)  
El cliente ('127.0.0.1', 41612) nos ha escrito: Conectado con el servidor  
El cliente ('127.0.0.1', 41612) nos ha escrito: Hola, soy cliente 2  
El cliente ('127.0.0.1', 41612) nos ha escrito: Hablamos?  
█
```

```
adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...  
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT  
/mi moodel/Python$ python3 'Prueba cliente tema 4.py'  
Socket cliente creado  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> Hola  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> Soy cliente 1  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> Me voy  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> bye  
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT  
/mi moodel/Python$ █
```

```
adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...  
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT  
/mi moodel/Python$ python3 'Prueba cliente tema 4.py'  
Socket cliente creado  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> Hola, soy cliente 2  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> Hablamos?  
Recibido del servidor:mensaje recibido  
Escribe tu mensaje (Para finalizar escribe: bye)--> █
```

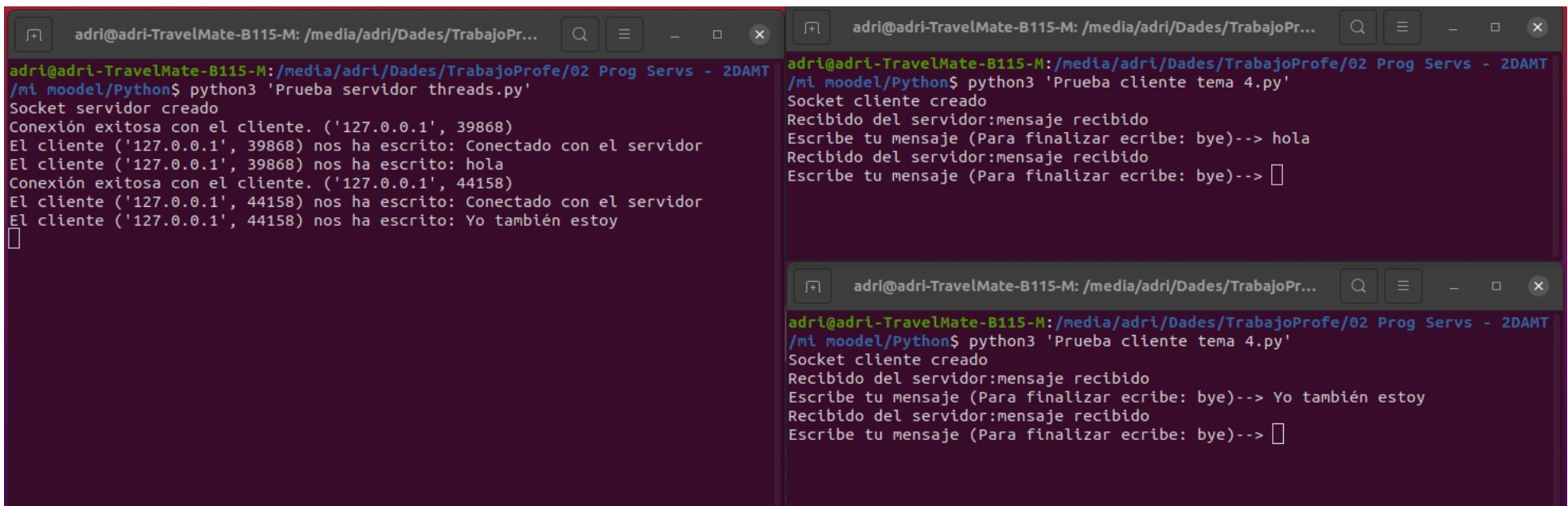

3.2. Problema a resolver con un ejercicio entregable

El servidor que os facilito, atiende un cliente y luego a otro y luego a otro. No puede atender a varios clientes a la vez.

Esta problemática se resuelve con el tema 1, creando hilos (Thread).

Si modificamos el programa de la parte del servidor, podremos atender a varios clientes al mismo tiempo.

Te dejo el ejemplo de la salida que puedes conseguir modificando el programa del servidor:



```
adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba servidor threads.py'
Socket servidor creado
Conexión exitosa con el cliente. ('127.0.0.1', 39868)
El cliente ('127.0.0.1', 39868) nos ha escrito: Conectado con el servidor
El cliente ('127.0.0.1', 39868) nos ha escrito: hola
Conexión exitosa con el cliente. ('127.0.0.1', 44158)
El cliente ('127.0.0.1', 44158) nos ha escrito: Conectado con el servidor
El cliente ('127.0.0.1', 44158) nos ha escrito: Yo también estoy
█

adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba cliente tema 4.py'
Socket cliente creado
Recibido del servidor:mensaje recibido
Escribe tu mensaje (Para finalizar escribe: bye)--> hola
Recibido del servidor:mensaje recibido
Escribe tu mensaje (Para finalizar escribe: bye)--> █

adri@adri-TravelMate-B115-M: /media/adri/Dades/TrabajoPr...
adri@adri-TravelMate-B115-M:/media/adri/Dades/TrabajoProfe/02 Prog Servs - 2DAMT
/mi moodel/Python$ python3 'Prueba cliente tema 4.py'
Socket cliente creado
Recibido del servidor:mensaje recibido
Escribe tu mensaje (Para finalizar escribe: bye)--> Yo también estoy
Recibido del servidor:mensaje recibido
Escribe tu mensaje (Para finalizar escribe: bye)--> █
```