PRACTICA UDP

OBJETIVO:

El objetivo de esta práctica es:

Comprender la comunicación ente procesos mediante el uso de Datagramas UDP. Para ello, Python proporciona los recursos necesarios en la librería socket.

A continuación, se proporcionan un código muy sencillo para un cliente:

```python
import socket

msg = "Hello Server"
bytes_tx= str.encode(msg)

server_address    = ("127.0.0.1", 6780)
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
socket.sendto(bytes_tx, server_address)

bytes_rx = socket.recvfrom(1024)
print("RX: ",bytes_rx)
socket.close()
```

Y para un servidor:

```python
import socket

msg = "Hello Client"
bytes_tx= str.encode(msg)

socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
socket.bind(("127.0.0.1", 6780))

bytes_rx = socket.recvfrom(1024)
message = bytes_rx[0]
address = bytes_rx[1]
print(message)
socket.sendto(bytes_tx, address)
```

Ejecútalos en terminales o en notebooks diferentes para comprobar como el servidor recibe un mensaje y envía una respuesta que es capturada por el cliente.

Intenta entender todas las sentencias de Python antes de ponerte con el resto de la práctica.

PRACTICA UDP 1:

Probar y analizar el funcionamiento de una sencilla aplicación cliente/servidor UDP en el que el servidor recibe un mensaje en el que el cliente se presenta y acto seguido le responde con un mensaje de bienvenida. Modifica el código proporcionado en el enunciado para conseguir que el servidor quede a la espera de un numero indefinido de clientes. Los clientes enviarán un mensaje del tipo: "**Hello Server, I'm Client 1**" y el servidor responderá con un "**Hello Client**"

- Para ello, se deberá arrancar primeramente el servidor desde un terminal o notebook, el cual quedará a la escucha de las conexiones de los clientes.
- Comprobar que cada cliente envía una única petición y finaliza. El servidor se quedará en espera por nuevas peticiones. Realizar la prueba de ejecución para dos clientes, desde dos terminales (o desde dos notebooks) en la misma máquina donde se ejecuta el servidor.
- Realizar igualmente la prueba de ejecución para un cliente desde un terminal en otra máquina diferente a donde se ejecuta el servidor (ponerse de acuerdo con algún compañero o virtualizar una segunda máquina en el mismo equipo).

PRACTICA UDP 2:

Modificad el programa ***servidor*** anterior:

- El código Python debe pedir al usuario el *número de puerto* (bien conocido) a utilizar en el servicio. Utilizar la instrucción ***input()***

  Por ejemplo:
  > nombre = input("What's your name")

- El código Python debe chequear que el parámetro introducido es efectivamente un número y que el servicio de escucha se levanta correctamente. Utilizar para ello las funciones de conversión de tipo de datos ***(str, int, float…)*** y la captura de excepciones ***try/except***

  Por ejemplo:

```
1 try:
2     print(variable)
3 except:
4     print("something happened on the print(variable) statement")
```

- Mejora el código del cliente haciendo uso de sentencias ***print()*** para imprimir mensajes informativos cuando el servidor está esperando una conexión, cuando se conecta un cliente, imprimiendo la dirección del cliente que le envía el mensaje además del mensaje recibido, …. Por ejemplo:

```
Provide port number (e.g. 6789): ABCD
It has to be a number (e.g. 6789)
Provide port number (e.g. 6789): 6789
UDP server up and listening
Message from Client: b"Hello UDP Server I'm client 1"
Client IP Address: ('127.0.0.1', 58576)
Message from Client: b"Hello UDP Server I'm client 2"
Client IP Address: ('127.0.0.1', 58591)
```

Modificad el programa ***cliente*** anterior para aplicar las mismas mejoras que en el servidor.

PRACTICA UDP 3:

Modificad el programa ***cliente*** anterior:

- Para que se envíe al servidor, los mensajes que introduce un usuario por la entrada estándar (usar el comando ***input***)
- Identificar a un cliente con una variable, por ejemplo, ***clientid=1***
- Utiliza la librería **pickle** para empaquetar ambos argumentos (mensaje e identificador del cliente) en una única secuencia de bytes
- Enviar al servidor tanto el mensaje como el identificador del cliente empaquetado
- Quedar a la espera de un mensaje del servidor informando de la recepción del mensaje y del número de caracteres recibidos

Por ejemplo:

```
Introduce message to send: Hi server, this is a test message
Message sent to Server
...
Message received from Server:  ('127.0.0.1', 6789)
Message:  Received 33 characters from client 1
```

Modificad el programa ***servidor*** anterior:

- Para que reciba los mensajes serializados con **pickle** y desempaquete el mensaje y el identificador del correo
- Utilice el identificador del cliente para mostrar un mensaje informativo sobre el cliente conectado
- Imprima la dirección y el puerto desde el que conecta el cliente
- Cuente el número de caracteres en el mensaje enviado
- Envíe al cliente un mensaje indicando que ha recibido un mensaje con el número de caracteres

Por ejemplo:

```
UDP server up and listening
Message received client  1
Message received from  ('127.0.0.1', 50511)
Number of characters in message:  23
Message received client  2
Message received from  ('127.0.0.1', 50514)
Number of characters in message:  38
Message received client  3
Message received from  ('127.0.0.1', 50533)
Number of characters in message:  39
```

PRACTICA UDP 4:

Modificar el código anterior para que en caso de que el servidor no esté arrancado, el cliente se quede esperando un tiempo determinado, por ejemplo 10 segundos. Transcurrido ese tiempo, si el servidor sigue caído, imprime un mensaje y cierra el socket. Además:

- Modificar el código del cliente y del servidor para que el usuario pueda pasar como argumentos de la línea de comandos la IP del servidor, el puerto al que se conecta y en el caso del cliente, el identificador del mismo. Los parámetros pasados en línea de comando se reciben en Python en la variable ***argv***. Por ejemplo:

  Un fichero "script.py" como este:

```
from sys import argv

print(argv[0])
print(argv[1])
print(argv[2])
```

  $ python script.py ARG1 ARG2 ARG3
  script.py
  ARG1
  ARG2

- El código de ambos debe seguir verificando que el número de puerto es numérico.
- El código del cliente debe poder enviar tantos mensajes como quiera, hasta que el usuario envíe la palabra "**EXIT**" que significará el fin de la ejecución del cliente y del servidor.
- El código del servidor, para evitar que se quede esperando de forma indefinida, debe poder finalizar al recibir un mensaje con la palabra "**EXIT**" por parte de cualquier cliente. Antes de terminar, enviará un mensaje de despedida al cliente que le envió la orden de terminar.

Por ejemplo, para el cliente:

```
Message> Hola, hay alguien ahí?
Waiting for server...
Waiting for server...
Waiting for server...
Message received from Server:  ('127.0.0.1', 6789)
Message:  Received 22 characters from client 1
Message> Buenos días, veo que ya estas funcionando
Message received from Server:  ('127.0.0.1', 6789)
Message:  Received 41 characters from client 1
Message> Me alegro, ahora soy yo quien se va
Message received from Server:  ('127.0.0.1', 6789)
Message:  Received 35 characters from client 1
Message> EXIT
Message received from Server:  ('127.0.0.1', 6789)
Message:  Bye Client 1
```

Y para el servidor:

```
UDP server up and listening
Message received client  1
Message received from  ('127.0.0.1', 59647)
Number of characters in message:  22
Message received client  1
Message received from  ('127.0.0.1', 59647)
Number of characters in message:  41
Message received client  1
Message received from  ('127.0.0.1', 59647)
Number of characters in message:  35
Message received client  1
Message received from  ('127.0.0.1', 59647)
Number of characters in message:  4
Received EXIT command from client 1
Bye
```

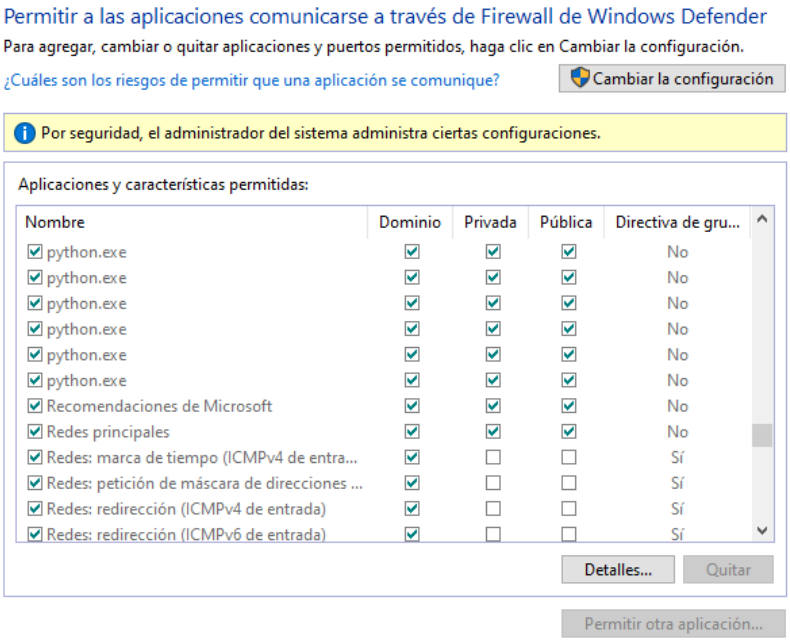
PRACTICA UDP 5:

Se ha de crear una aplicación cliente/servidor, donde el servidor proporcionará la hora y el día a los clientes que lo soliciten.
El programa cliente realizará una petición al servidor y esperará la respuesta un tiempo limitado de 5000 milisegundos. Intentar implementar esta funcionalidad capturando excepciones y usando la función ***sleep*** del paquete de Python ***time***.
Si recibe la respuesta, el cliente presentará por pantalla el día y la hora enviados por el servidor. Si pasado el tiempo máximo de espera no recibe una respuesta, enviará a la pantalla un mensaje de error diciendo que el servidor no está online.

PRACTICA UDP 6:

Modificar el cliente de la práctica anterior para que compare la hora local con la hora remota enviada por el servidor indique, con un mensaje por pantalla, la diferencia entre ellas. Probarlo ejecutando dos procesos en dos máquinas diferentes para que la diferencia sea medible.

Para que la diferencia de tiempos sea medible, habría que ejecutarlo en dos equipos independientes conectados por red. Para lo cual la red no debería contar con las políticas de restricción de la universidad. Igualmente, es posible que en los portátiles haya que acceder al firewall para asignar permisos absolutos al ejecutable de Python:



PRACTICA UDP ENTREGABLE:

Vamos a implementar un módulo para jugar online. El juego que se propone es muy sencillo. Es un juego de preguntas y respuestas. Podéis utilizar algunas APIs públicas y gratuitas para recibir preguntas:

- https://opentdb.com/
- https://the-trivia-api.com/
- https://api-ninjas.com/api/trivia

El servidor debe enviar las preguntas y las posibles respuestas (en caso de ser un juego de respuesta múltiple) a los jugadores, que devolverán al servidor la respuesta. El servidor validará si es un acierto o no y e informará al usuario del resultado. Todas estas comunicaciones deben realizarse mediante mensajes UDP.

ENTREGA:

Se deberá entregar un documento donde se recojan y analicen los siguientes puntos:

a) Describir brevemente las partes del código e instrucciones para ejecutarlo

b) Capturas de pantalla del juego

c) Código de la práctica planteada