PRACTICA TCP

OBJETIVO:

El objetivo de esta práctica es comprender la comunicación ente diferentes procesos mediante TCP. A continuación, se proporcionan un par de códigos muy sencillos para una comunicación cliente/servidor.

Para el cliente:

```python
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 6780)
sock.connect(server_address)
sock.sendall("Hi TCP Server".encode())
data=sock.recv(1024).decode()
print("RX: ",data)
sock.close()
```

Y para el servidor:

```python
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 6780)
sock.bind(server_address)
sock.listen(1)
connection, client_address = sock.accept()
data = connection.recv(1024).decode()
print("RX: ",data)
connection.sendall("Hi TCP Client".encode())
connection.close()
```

Ejecutarlos 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 TCP 1:

Probar y analizar el funcionamiento de una sencilla aplicación cliente/servidor TCP donde el

servidor recibe un mensaje en el que el cliente se presenta y acto seguido el servidor le responde con un mensaje de bienvenida. El servidor debe permanecer a la escucha de más clientes hasta que reciba un mensaje de “EXIT” de alguno de ellos, en este caso, el servidor envía un mensaje de despedida “Bye” y se desconecta.

- Para ello, se deberá arrancar previamente el servidor desde un terminal (o notebook), el cual se quedará a la escucha de las peticiones de los clientes.
- Comprobar que cada cliente envía una única petición y finaliza. El servidor se quedará en espera de nuevas peticiones hasta que uno de esos mensajes sea "EXIT".
- 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 la prueba de ejecución para un cliente localizado en otra máquina diferente a aquella en donde se ejecuta el servidor (ponerse de acuerdo con algún compañero o virtualizar una segunda máquina en el mismo equipo, pero con otra IP).

Esta podría ser la salida del servidor:

```
Waiting for connections...
RX:  Hello Server, I'm Client 1
Waiting for connections...
RX:  Hello Server, I'm Client 2
Waiting for connections...
RX:  Hello Server, I'm Client 3
Waiting for connections...
RX:  EXIT
Bye
```

PRACTICA TCP 2:

Modificar el programa anterior para simular el envío de paquetes de un tamaño superior al buffer de recepción. Para el envío no hay problemas ya que la instrucción ***sendall*** se encarga de enviar tantos paquetes como sea necesario. Sin embargo, no existe una instrucción ***recvall***. Vamos a forzar que tanto el cliente como el servidor sólo puedan recibir paquetes de un máximo de 20 bytes con la instrucción ***socket.recv(20)***. Esto implica que el código Python debe iterar hasta que se ha recibido el mensaje completo. Algo similar a esto:

```
while True:
    data = sock.recv(MTU).decode()
    if len(data)==MTU:
        print("RX: ",data)
    else:
        print("RX: ",data)
        break
```

- El servidor deberá estar escuchando constantemente conexiones de clientes
- Los clientes enviarán mensajes largos del tipo: "Hello Server, This is a long message that you will received in chunks of 20 bytes"
- En cuanto se haya establecido una conexión, el servidor recibirá los mensajes en ***chunks*** de 20 bytes y los irá imprimiendo según los vaya recibiendo

- Cuando el servidor haya recibido un mensaje completo del cliente (cuando el último ***chunk*** recibido no llegue a 20 bytes), el servidor enviará una respuesta que también superará con mucho los 20 bytes: "Hello Client, this is a long message that you will received in chunks of 20 bytes"
- Después de haber enviado ese mensaje, el servidor seguirá esperando por nuevos clientes
- El cliente recibirá el mensaje largo enviado por el servidor y lo imprimirá en tantos ***chunks*** como sea necesario. En cuanto haya recibido el mensaje completo (el último ***chunk*** no llegue a 20 bytes), el cliente se desconectará
- El servidor seguirá funcionando esperando nuevas conexiones hasta que reciba un mensaje de un cliente con el texto “EXIT”

Esta podría ser la salida de un cliente:

```
Connecting to server ('localhost', 6780)
RX:  Hello Client, this i
RX:  s a long message tha
RX:  t you will received
RX:  in chunks of 20 byte
RX:  s
Closing socket
```

Y esta la del servidor después de procesar un par de clientes:

```
Waiting for connections
Connection from ('127.0.0.1', 53801)
RX: Hello Server, This i
RX: s a long message tha
RX: t you will received
RX: in chunks of 20 byte
RX: s
Waiting for connections
Connection from ('127.0.0.1', 53808)
RX: Hello Server, This i
RX: s a long message tha
RX: t you will received
RX: in chunks of 20 byte
RX: s
Waiting for connections
Connection from ('127.0.0.1', 53828)
RX: EXIT
Bye
```

PRACTICA TCP 3:

Codificar un programa cliente/servidor que permita la recepción desde el servidor, de un fichero solicitado por un cliente. El servidor aceptará el siguiente formato de peticiones:

1. Cuando el cliente quiera un fichero del servidor, utilizará el mensaje: ***get "nombre_fichero"***

    a) Si el fichero existe, el servidor se lo enviará al cliente

    b) Si el fichero no existe, el servidor enviará un mensaje avisando de esta circunstancia.

2. Este servicio dispondrá de un comando "***put***" que permitirá al cliente enviar ficheros al servidor: ***put "nombre_fichero"***

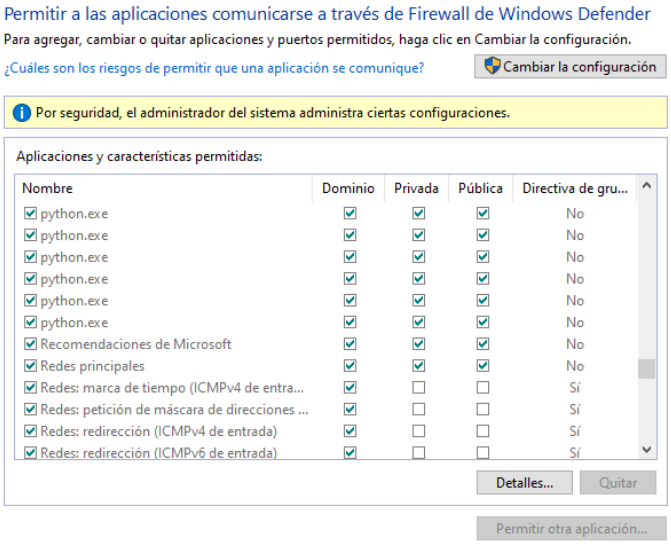
3. Este servicio cliente-servidor dispondrá de la posibilidad de enviar un fichero al servidor de forma exclusiva por un único cliente con la intención de modificarlo. Para ello, el cliente tiene que solicitar previamente el bloqueo del fichero. El servidor realizará este bloqueo identificando al cliente por su dirección IP y puerto de escucha. Una vez que el servidor ha bloqueado el fichero con la información del cliente, el cliente puede proceder a descargarlo y modificarlo.

    Tras la modificación de dicho fichero (en el cliente), este podrá enviar al servidor el contenido del nuevo fichero y actualizar el fichero original (alojado en el servidor). Para ello, se deberá seguir el siguiente protocolo:

    3.1. Bloqueo del fichero. El cliente ejecuta el comando: ***lock "nombre_fichero"***
    3.2. El servidor bloquea el fichero utilizando el ID, IP y Puerto del cliente.
    3.3. El cliente descarga el fichero con el comando: **get "nombre_fichero"**
    3.4. El cliente modifica el fichero
    3.5. El cliente envía el fichero modificado al servidor con el comando: ***put "nombre_fichero"***
    3.6. El servidor actualiza el fichero con el nuevo contenido
    3.7. El cliente solicita el desbloqueo del fichero en el servidor con el comando: ***unlock "nombre_fichero"***

NOTA1: Si se utiliza el mismo ordenador para implementar el cliente y el servidor, el fichero con el proceso cliente debe estar situado en una carpeta distinta al proceso server para poder observar el correcto envío de fichero en localizaciones diferentes. O bien, tomar la precaución de renombrar el fichero antes de crearlo.

NOTA2. Recuerda que para comunicar dos equipos que se encuentren en la misma red o en redes diferentes, hay que asegurarse que no existe ninguna política que bloquea ese tipo de comunicación. Igualmente, es posible que en los portátiles haya que acceder al firewall para asignar permisos absolutos al ejecutable de Python.



PRACTICA TCP ENTREGABLE:

Vamos a implementar un módulo de registro y login de jugadores para un servidor de juegos online. El servidor dispone de un fichero con la lista de usuarios y contraseñas que están autorizados a conectarse para jugar.

El módulo debe permitir:

1. Que un nuevo jugador se registre. El registro sólo implica el envío por parte del nuevo jugador de su username y de su password. El servidor debe comprobar que el username no existe ya en su base de datos. No existen normas respecto a la complejidad del password.
2. Que un jugador haga loggin. Este proceso se realiza enviando al servidor login y password. El servidor validará que dicha pareja de variables existe en su fichero de credenciales y devolverá un mensaje con el resultado al cliente.
3. En caso de autenticación exitosa, el servidor mantendrá la conexión con todos ellos y registrará en alguna estructura de datos quienes son y cuál es su socket para comunicarse con ellos. En caso de autenticación fallida, el servidor cerrará el canal de comunicación con el cliente.

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