PRACTICA Multithreading

OBJETIVO:

Mejorar los mecanismos de comunicación cliente/servidor implementados en las prácticas utilizando multithreading.

PRACTICA Multithreading 1.1:

Implementar un mecanismo de comunicación cliente servidor con las siguientes características.

Servidor:

- Se debe solicitar al usuario el puerto en el que levantar el servicio y el código debe asegurarse que el puerto es válido
- Se creará un servicio de escucha con un socket utilizando paquetes TCP
- Cada vez que el servidor reciba una conexión de un cliente, debe crear una hebra para tratar esa conexión. La hebra debe recibir los datos, imprimirlos por pantalla, devolverlos al cliente a modo de eco y cerrar la conexión antes de finalizar su ejecución
- El servidor quedará escuchando nuevos clientes

Cliente:

- Se preguntará al usuario la dirección del servidor al que conectarse
- Se preguntará al usuario el puerto al que enviar los mensajes
- Se pedirá al usuario que introduzca varios mensajes que irán almacenándose en una lista. El último mensaje será siempre "***exit***"
- Para emular varios clientes conectando simultáneamente con el servidor se lanzará una hebra por mensaje introducido por el usuario. Cada hebra debe crear una conexión con el servidor, enviar el mensaje, esperar la respuesta y cerrar el socket antes de finalizar.

NOTA. Dependiendo de como realice la gestión de los recursos compartidos el S.O. este efecto puede ser más o menos notorio. Se puede forzar este efecto haciendo que el cliente en lugar de llamar una sola vez al comando PRINT con el mensaje recibido del servidor lo llame una vez con cada palabra del mensaje (usando un bucle FOR y el comando SPLIT); de esta manera se apreciará mejor los cambios de contexto por las hebras.

ENTREGA:

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

a) Describir brevemente las partes del código.

b) Resultados obtenidos de la ejecución.

c) Código de la práctica planteada.

PRACTICA Multithreading 1.2:

De la ejecución del módulo anterior, se observa que la ejecución concurrente de hebras puede dar lugar a salidas como las siguientes:

```
Introduce la dirección del servidor
localhost
Introduce el puerto de escucha del servidor
6780
Introduce los mensajes a enviar, para terminar escribir exit
Hi, this is first message from Client 1
Introduce los mensajes a enviar, para terminar escribir exit
AND THIS IS SECOND MESSAGE FROM CLIENT 2
Introduce los mensajes a enviar, para terminar escribir exit
Hola, este es el tercer mensaje del cliente 3
Introduce los mensajes a enviar, para terminar escribir exit
Y ESTE ES EL CUARTO MENSAJE DEL CLIENTE 4
Introduce los mensajes a enviar, para terminar escribir exit
exit
12 : sending  AND THIS IS SECOND MESSAGE FROM CLIENT 2
 : sending  Hi, this is first message from Client 1
3 : sending  Hola, este es el tercer mensaje del cliente 3
4 : sending  Y ESTE ES EL CUARTO MENSAJE DEL CLIENTE 4
12 : received from the server : AND THIS IS SECOND MESSAGE FROM CLIENT 2
2 : closing socket
 : received from the server : Hi, this is first message from Client 1
1 : closing socket
43 : received from the server : Y ESTE ES EL CUARTO MENSAJE DEL CLIENTE 4
4 : closing socket
 : received from the server : Hola, este es el tercer mensaje del cliente 3
3 : closing socket
```

De esta salida se percibe que la impresión por pantalla se ve afectada por la propia concurrencia de las hebras. Para evitarlo, se pueden usar ***locks***. Mirar el apartado "*Synchronizing Threads*" del siguiente link:

https://www.tutorialspoint.com/python3/python_multithreading.htm

Mejorar el código del cliente y del servidor implementando el mecanismo descrito. Para ello crear un ***lock*** como una variable global en el programa principal:

```
print_lock = threading.Lock()
```

Las hebras adquirirán el ***lock*** cada vez que necesiten hacer uso de la impresión por pantalla y lo liberarán cuando ya no lo necesiten.

ENTREGA:

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

a) Describir brevemente las partes del código.

b) Resultados obtenidos de la ejecución.

c) Código y las modificaciones que resuelvan la práctica planteada.

La salida de ambos módulos, usando ***locks***, debería ser similar a esta:

Emulador de clientes:

```
Introduce el puerto de escucha
6780
starting up on port ('localhost', 6780)
socket is listening
waiting for a connection
Connected to : 127.0.0.1 : 50210
waiting for a connection
Connected to : 127.0.0.1 : 50211
waiting for a connection
Connected to : 127.0.0.1 : 50216
waiting for a connection
received: -> Hi, this is message 1 from client1
Connected to : 127.0.0.1 : 50217
waiting for a connection
sending data back to the client
received: -> HI THIS IS MESSAGE 2 FROM CLIENTE 2
sending data back to the client
received: -> Hola, este es el mensaje 3 del cliente 3
sending data back to the client
received: -> HOLA, ESTE ES EL MENSAJE 4 DEL CLIENTE 4
sending data back to the client
```

Servidor:

```
Introduce la dirección del servidor
localhost
Introduce el puerto de escucha del servidor
6780
Introduce los mensajes a enviar, para terminar escribir exit
Hi, this is message 1 from client1
Introduce los mensajes a enviar, para terminar escribir exit
HI THIS IS MESSAGE 2 FROM CLIENTE 2
Introduce los mensajes a enviar, para terminar escribir exit
Hola, este es el mensaje 3 del cliente 3
Introduce los mensajes a enviar, para terminar escribir exit
HOLA, ESTE ES EL MENSAJE 4 DEL CLIENTE 4
Introduce los mensajes a enviar, para terminar escribir exit
exit
2 : sending  HI THIS IS MESSAGE 2 FROM CLIENTE 2
1 : sending  Hi, this is message 1 from client1
3 : sending  Hola, este es el mensaje 3 del cliente 3
4 : sending  HOLA, ESTE ES EL MENSAJE 4 DEL CLIENTE 4
1 : received from the server : Hi, this is message 1 from client1
1 : closing socket
2 : received from the server : HI THIS IS MESSAGE 2 FROM CLIENTE 2
2 : closing socket
3 : received from the server : Hola, este es el mensaje 3 del cliente 3
3 : closing socket
4 : received from the server : HOLA, ESTE ES EL MENSAJE 4 DEL CLIENTE 4
4 : closing socket
```