Sockets TCP en Python

1. Introducción

Las prácticas de la asignatura Internet Redes y Datos constarán de un conjunto de ejercicios **no obligatorios**. Todas las prácticas han de ser realizadas de manera **individual**.

Cada práctica constará de un enunciado donde se plantearán las tareas a llevar a cabo, y podrá incluir información o referencias complementarias para la realización de las mismas. Como ejemplo, se recomienda consultar para cualquier duda relacionada con el lenguaje de programación Python o sus librerías los recursos disponibles en: https://docs.python.org/3/.

Esta práctica proporciona una introducción a la programación con sockets TCP en Python y presenta la primera práctica puntuable de la asignatura.

Esta parte será evaluada y corregida como se indica en el apartado 4.

2. Conceptos básicos

2.1. Repaso TCP/UDP

La pila de protocolos TCP/IP está formada por los niveles de Aplicación, Transporte, Red y Acceso al medio. De manera habitual, cuando se escriben aplicaciones en red, se trabajará a nivel de aplicación y se utilizarán además protocolos del nivel de transporte.

Las principales diferencias entre los protocolos TCP y UDP del nivel de transporte son los siguientes:

- TCP (Transmission Control Protocol):
  - Protocolo orientado a conexión.
  - Provee un flujo de bytes fiable.
  - Protocolos a nivel de aplicación: telnet, HTTP, FTP, SMTP, ...
- UDP (User Datagram Protocol):
  - Protocolo no orientado a conexión.
  - Envía paquetes de datos (datagramas) independientes y sin garantías.
  - Permite broadcast y multicast.
  - Protocolos a nivel de aplicación: DNS, TFTP, ...

2.2. Sockets

Cuando trabajamos en una red de ordenadores y queremos establecer una comunicación (enviar o recibir datos) entre dos procesos que se están ejecutando en máquinas diferentes de dicha red necesitamos hacer uso de los *Sockets* para abstraer las conexiones.

Para diferenciar estas conexiones se les asigna un número de 16 *bits* (entre 0 y 65535) conocido como puerto. Por tanto para dirigir de manera unívoca la información entre dos procesos en diferentes máquinas hemos de hacer uso de la dirección IP y el puerto asignado en cada máquina, así como del protocolo empleado.

Según el papel que cumplan los programas podremos hablar de cliente o servidor, siendo el primero el que inicia una petición y recibe la respuesta y el segundo el que se encuentra a la espera de recibir una conexión para generar una respuesta.

Se puede consultar la documentación relativa a Sockets en Python en el siguiente enlace:

https://docs.python.org/3/library/socket.html

2.3. Comandos

Resulta útil hacer uso de un comando para la prueba de los programas que vamos a desarrollar en las prácticas, es el *nc* o *Netcat*. Este programa nos permite enviar datos a través de una red, indicando el destino y el protocolo a emplear, entre otras opciones.

Especialmente útiles pueden resultar las siguientes combinaciones:

```
# Abrir un servidor TCP escuchando en un determinado puerto
> nc -t -l -p <puerto>
# Abrir un cliente TCP para enviar un mensaje
> nc -t <ip> <puerto>
```

3. Ejercicios

3.1. Sockets TCP

Los *sockets* TCP son orientados a conexión y fiables, esto implica que antes de poder enviar o recibir datos es necesario establecer una conexión entre el cliente y el servidor. Una vez esta conexión está establecida el protocolo garantiza que los datos son recibidos de manera correcta y en orden en el destino.

3.1.1. Ejercicio 1: Cliente TCP

1. Se crea un *socket* orientado a conexión, usando TCP.
2. Establecemos un *Timeout* para el *socket*, de 300 segundos.

3. Conectamos el *socket* con el servidor mediante la función *connect*, indicando la dirección (máquina y puerto).
4. Enviamos el mensaje codificado a la dirección destino, representada por una tupla (maquina, puerto) mediante la función *send*.
5. Recibimos la respuesta en el mismo *socket* abierto (*recv*).
6. Decodificamos y mostramos el mensaje recibido.
7. Controlamos la excepción por *timeout (socket.timeout)* y cualquier otra excepción posible.
8. En cualquier caso tratamos de cerrar el *socket* al final de la ejecución.

3.1.2. Ejercicio 2: Servidor TCP

Implementaremos ahora un servidor eco TCP. Para ello se seguirán los siguientes pasos:

1. Creamos un *socket* orientado a conexión.
2. Asociamos el *socket* con una dirección y puerto mediante el método *bind*.
3. Establecemos un *timeout* para el *socket*, de 300 segundos.
4. Ponemos el servidor en modo escucha mediante el método *listen()*.
5. Creamos un bucle infinito
   a) Invocamos el método *accept* que se queda esperando hasta recibir la petición de conexión de un cliente. Cuando se produce la conexión, devuelve un nuevo *socket* que es el utilizado para la comunicación con ese cliente.
   b) Recibimos el mensaje en el *socket* del cliente.
   c) Mostramos el contenido del mensaje.
   d) Enviamos el mismo mensaje recibido a través del *socket* del cliente.
   e) Cerramos el *socket* del cliente.
6. Controlamos la excepción por *timeout (socket.timeout)* y cualquier otra excepción posible.
7. Cerramos el *socket* del servidor al final de la ejecución.

3.1.3. Ejercicio 3: Servidor TCP multihilo

Los servidores TCP son habitualmente multihilo, para poder procesar más de una petición de manera simultánea. Para esto han de crearse hilos de ejecución a partir del programa principal, que serán los que se ocuparán de las peticiones de cada cliente.

Para ello, partiendo del servidor TCP implementado en el ejercicio anterior, crearemos hilos de ejecución mediante la función *Thread* del módulo *threading*. De esta manera los apartados *b, c, d* y *e* del punto *4* pasarán a estar definidas dentro de una función. Esta será la que se le proporcione a la función de creación de hilos de la siguiente manera:

```
threading.Thread(target=NOMBRE_DE_LA_FUNCION,
                 args=(ARGUMENTOS_DE_LA_FUNCION))
                 .start()
```

Una vez creado el hilo se debe lanzar su ejecución (mediante el método *start()*), quedando el resto del código del programa igual al del ejercicio anterior.

Para comprobar el correcto funcionamiento de este apartado deberán seguirse los siguientes pasos:

1. Ejecutar servidor multihilo TCP.
2. Abrir una conexión con netcat contra el servidor:

   ```
   nc -t <ip> <puerto>
   ```

3. Ejecutar el cliente TCP.

Si al ejecutar estos pasos el cliente recibe respuesta del servidor, este tendrá la función multihilo implementada de manera correcta, ya que es capaz de responder a una petición mientras tiene otra abierta.

4. Evaluación y pruebas

Para la evaluación de esta práctica se requerirá haber realizado **al menos un *commit* por cada ejercicio propuesto** (i.e. 3 *commits*). La puntuación correspondiente a esta práctica es de **0.25** sobre la nota final, repartiéndose entre los tres ejercicios y su correcta defensa.

La entrega se realizará en el repositorio Git, siendo la **fecha límite las 23:59 del 26/02/2021 (CET o UTC+01:00)**. Las defensas se realizarán en la semana del 1 al 5 de marzo.

Adicionalmente, esta práctica podría realizarse y defenderse en un momento anterior a la fecha de la entrega con acuerdo previo con el profesor de prácticas.