Programando en Python con Sockets

Los sockets permiten comunicar distintos nodos a través de una red. En esta actividad vamos a desarrollar un cliente y un servidor en Python, y haremos que se envíen mensajes a través de sockets.

Los sockets se usan para transmitir mensajes entre dispositivos sobre redes locales o globales. Un socket es un extremo de una comunicación y es por eso que los sockets son útiles en aplicaciones cliente-servidor de tiempo real sobre pasaje de mensajes, como WhatsApp.

> Programar con sockets nos permite comunicar nodos sobre una red. Los nodos pueden ser **clientes** y **servidores**. Un **servidor** es un programa de software que espera *requests* de clientes y atiende los procesos entrantes, mientras que el **cliente** es quien solicita los servicios. Un cliente solicita recursos del servidor y el servidor responde a la solicitud.

Por ejemplo: para leer un artículo en Internet, tu navegador establece una conexión por sockets al servidor del sitio web correspondiente y envía un mensaje *request* al servidor, solicitando el artículo que se quiere leer. El servidor entonces envía el artículo a tu navegador a través de la conexión por sockets. Así, los sockets permiten establecer una conexión entre tu navegador y el servidor del sitio web.

En esta actividad, vas a hacer un poco de *socket programming* en Python. Para eso, vas a crear aplicaciones servidor y cliente basados en sockets usando Python.

¿Qué es *socket programming*?

Socket programming es una forma de comunicación entre nodos sobre una red. Los nodos pueden ser clientes y servidores. Un servidor es un programa de software que espera requests de clientes y atiende los procesos entrantes, mientras que el cliente es quien solicita los servicios. Un cliente solicita recursos del servidor y el servidor responde a la solicitud.

Para establecer una conexión entre un cliente y un servidor, creamos un socket en cada programa y los conectamos. Una vez que se estableció la conexión entre los sockets, los nodos pueden intercambiar datos.

El módulo Socket() en Python

En Python, el módulo *socket* provee una forma de trabajar con sockets. Como ya venimos diciendo, los sockets nos permiten establecer conexiones de red mediante varios protocolos, como TCP o UDP, para enviar y recibir datos. Para usarlo en tu código, primero tenés que importar el módulo *socket*.

Una vez que importaste el módulo, ya podés usar el método `socket()` para crear un objeto socket. Acá hay una lista de algunos de los métodos más importantes de este módulo:

Métodos de Socket	Descripción
socket.socket()	Crea un objeto socket nuevo solicitando recursos al sistema operativo.
.bind()	Asocia un socket con una dirección IP y un puerto específicos.
.listen()	Pone al servidor en modo listening para que pueda escuchar conexiones entrantes.
.accept()	Acepta conexiones entrantes de un cliente en un socket del servidor.
.connect()	Conecta un socket de un cliente a un servidor remoto.
.send()	Envía datos a los endpoints conectados. Lo usan tanto cliente como servidor.
.recv()	Usado tanto por cliente como por servidor para recibir datos del otro.
.close()	Usado para cerrar y terminar la conexión del socket y liberar todos los recursos utilizados.

Entendiendo el modelo cliente-servidor

Ya sabemos que los clientes solicitan algunos recursos al socket del servidor, y el servidor responde a esa petición. Ahora vamos a ver cómo codear tanto al servidor como al cliente para que puedan comunicarse entre sí.

Servidor

El servidor ejecuta primero y espera cualquier *request* de clientes. El servidor tiene el método `bind()` que asocia la dirección IP y el número de puerto, para que pueda escuchar *requests*

entrantes de clientes en esa IP y ese puerto. En el método bind, en lugar de poner una IP vamos a pasar un string vacío, esto hará que el servidor escuche *requests* que lleguen de otras computadoras de la red.

Luego, el servidor tiene el método `listen()`, que pone al servidor en modo listening, así puede escuchar las conexiones entrantes. Por último, el servidor tiene un método `accept()` que inicia una conexión con el cliente, y finalmente el método `close()` que cierra la comunicación con el cliente.

Así que la secuencia es:

bind → listen → accept → comunicarse → close

Creando un Server Socket

Para crear un *server socket* hay que seguir estos pasos:

1. Importar el módulo socket (ya lo hicimos).
2. Obtener el *hostname* usando el método `elSocket.gethostname()`. Esto es opcional porque también podés querer pasar el string vacío para que el servidor pueda escuchar conexiones de otras computadoras.
3. Especificar un puerto en el que escuchar, teniendo en mente que el número de puerto debe ser más grande que 1024. Por ejemplo, podés poner 21042.
4. Crear un objeto socket usando el método `socket.socket()`.
5. *Bindear* el host y el número de puerto usando el método `bind()`.
6. Llamar al método `listen()`, podés configurar cuántos clientes puede escuchar el servidor simultáneamente.
7. Establecer una conexión con el cliente usando el método `accept()`.
8. Enviar el mensaje al cliente usando el método `send()`.
9. Cerrar la conexión con el cliente usando el método `close()`.

Cliente

El programa del cliente va a iniciar una conversación. Esto es muy similar al caso del servidor. La única diferencia es que en el programa del servidor usamos el método `bind()` para asociar la dirección y el puerto del *host*, mientras que en el programa del cliente vamos a usar el método `connect()` para conectar la dirección y el puerto del *host* así puede comunicarse con el servidor.

Creando un *Client Socket*

Para crear un *client socket*, seguir los siguientes pasos:

1. Importar el módulo socket.
2. (Opcional) Obtener el *hostname* usando el método `socket.gethostname()`.
3. Crear un objeto socket usando el método `socket.socket()`.
4. Conectar al *local host* pasando el número de puerto y el *hostname* del servidor.
5. Enviar y recibir mensajes del servidor usando los métodos `send()` y `recv()`.
6. Cerrar la conexión con el servidor.