Sockets

Antonio Espín Herranz

Sockets

- Python proporciona soporte para establecer comunicaciones entre máquinas.

- Un **socket** representa un punto de enchufe a la red definido por una **dirección IP, puerto y el protocolo que utiliza**.

Clasificación de Socket

- Los sockets se clasifican en:
  - Sockets de flujo (socket.SOCK_STREAM)
  - Sockets de datagramas (socket.SOCK_DGRAM)

- Dependiendo de si el servicio utiliza **TCP**, que es orientado a conexión y fiable, o **UDP**, respectivamente.

- Según **Unix**:
  - socket.AF_UNIX: Basados en ficheros.
  - socket.AF_INET: Basados en redes.

Socket

- Para crear socket:
  - socket.socket()
    - Por defecto se utiliza la familia AF_INET y el tipo SOCK_STREAM.

- Tanto el cliente como el servidor necesitará un socket para comunicarse.

Servidor

- **Crear** el **socket** para el servidor.
  - **socket_s = socket.socket()**

- Indicar en qué puerto se va a mantener a la escucha nuestro servidor utilizando el método **bind**.

- Para sockets IP, el **argumento de bind es una tupla** que contiene el **host** y el **puerto**.

- El host se puede dejar vacío, indicando al método que puede utilizar cualquier nombre que esté disponible.
  - **socket_s.bind((“localhost”, 9999))**

- Tener en cuenta que el **puerto** debe ser **mayor de 1024** porque están reservados

Servidor

- Para aceptar las conexiones de los clientes:
  - Métodos: **listen** y **accept**.
  - **El método listen** requiere de un parámetro que indica el **número de conexiones máximas** que queremos aceptar; evidentemente, este valor debe ser al menos 1.
  - **El método accept** para aceptar peticiones.
    - Devuelve **un objeto socket** que representa la conexión del cliente **y una tupla que contiene el host y puerto de dicha conexión**.
- **Código**:
  - socket_s.listen(10)
  - socket_c, (host_c, puerto_c) = socket_s.accept()

Servidor

- Cuando se ha establecido la conexión podemos enviar y recibir información con las funciones:
  - **recv** y **send** para TCP.
  - **recvfrom** y **sendfrom** en UDP.

- El método **send** recibe los datos a enviar.
- El método **recv** toma como parámetro el número máximo de bytes a aceptar.

- Por último cerrar el socket: **close**()

Cliente

- El cliente es más sencillo.
- **Crear** un objeto **socket**, utilizar el método **connect** para conectarnos al servidor y utilizar los métodos **send** y **recv** .
- El método **connect** recibe una tupla con **host** y **puerto**, exactamente igual que bind.
- **CÓDIGO**:
  - socket_c = socket.socket()
  - socket_c.connect((“localhost”, 9999))
  - socket_c.send(“hola”)

Código

- **SERVIDOR**

```
import socket
s = socket.socket()
s.bind((“localhost”, 9999))
s.listen(1)
sc, addr = s.accept()
while True:
        recibido = sc.recv(1024)
        recibido=recibido.decode(‘utf-8’)
        if recibido == “quit”:
                break
        print (“Recibido:”, recibido)
        sc.send(recibido.encode(‘utf-8’))

print (“fin comunicación”)
sc.close()
s.close()
```

- **CLIENTE**

```
import socket
s = socket.socket()
s.connect((“localhost”, 9999))

while True:
        mensaje = input(“> “)
        s.send(mensaje.encode(‘utf-8’))
        recibido = s.recv(1024)
        recibido=recibido.decode(‘utf-8’)
        if mensaje == “quit”:
                break

print (“Recibido:”, recibido )
print (“adios” )
s.close()
```

Reutilización de puertos

- Si no salimos correctamente del servidor y hacemos close del socket se quedará bloqueado con lo cual tendremos que cambiar de puerto.

- Al crear el socket podemos indicar que queremos reutilizar el puerto:

```
s = socket.socket()
#Reutilizar el socket:
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
```