

Sockets con Python

Introducción

Los sockets son virtualizaciones de puntos terminales de una comunicación bidireccional; tanto dentro de procesos (IPC) entre procesos en la misma máquina o en máquinas diferentes. Los más comunes: TCP, UDP y Unix domain sockets (IPC dentro del kernel Unix / Linux)

Permiten establecer una comunicación cliente-servidor. El cliente inicia la comunicación; el servidor espera a que uno o más clientes inicien una comunicación.

Python incluye mediante el módulo `socket` una interfaz a sockets BSD implementados en el sistema operativo -de bajo nivel-

Métodos de socket

Los métodos más importantes de **socket**:

Para Socket servidor:

- **`bind()`**

Conecta un socket a una dirección formada por dirección de host + port

- **`listen()`**

Inicia el socket server para monitorear conexiones - recibe un parámetro "backlog" que es la cantidad de conexiones máximas pendientes (queued)

- **`accept()`**

Bloquea la ejecución del programa hasta recibir una conexión de un cliente

Socket cliente:

- **`connect()`, `connect_ex()`**

Inicia una conexión con un socket servidor

connect_ex() devuelve códigos de error equivalentes a la llamada de bajo nivel *connect()* de C - 0 = No hubo error

connect() dispara excepciones en caso de error

Métodos generales de socket:

- **send()**

Envía un mensaje TCP

- **sendto()**

Envía un mensaje UDP

- **recv()**

Recibe un mensaje TCP

- **recvfrom()**

Recibe un mensaje UDP

- **close()**

Cierra el socket

- **gethostname()**

Devuelve el host

Los nombres de los métodos coinciden con la implementación en C (que es la que se ejecuta en última instancia)

Además, Python incluye clases de nivel más alto que simplifican algunas operaciones de red: HTTP (`httplib`), FTP (`ftplib`), SMTP (`smtplib`), POP3 (`poplib`), Telnet (`telnetlib`), etc.
(<https://docs.python.org/3/library/internet.html>)

Características de un socket

Algunos términos clave para entender la documentación sobre sockets, y para instanciar un objeto de esta clase:

dominio / domain

La familia de protocolos usados como transporte. (AF_INET, PF_INET, PF_UNIX, PF_X25, etc.)

AF_INET: "address family, internet" - direcciones IP

PF_INET: "protocol family, internet" - cualquier elemento del protocolo (sockets, ports, etc.)

tipo / type

Qué tipo de comunicación se usa entre dos puntos; SOCK_STREAM para protocolos orientados a conexión (TCP) o SOCK_DGRAM (Para protocolos sin conexión (UDP)

protocolo

Variante del protocolo que se usa; usualmente cero

hostname / nombre de host

Identifica el host. Puede ser un string con una dirección IPv4 (ej. '192.168.1.2')

IPv6 (ej. '::5') o un nombre "www.servidor.com")

Si el string está vacío indica INADDR_ANY, "<broadcast>" indica INADDR_BROADCAST (ambos no soportados en IPv6)

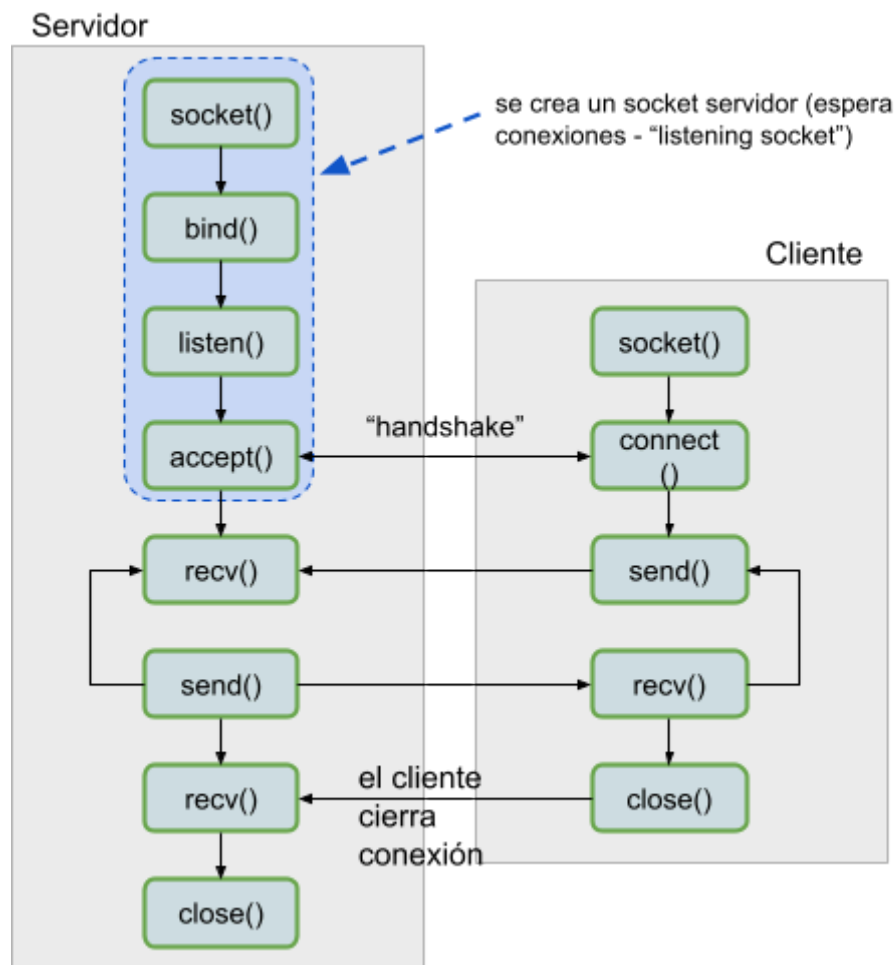
También se puede especificar como un entero.

port / puerto

El puerto que usa el servidor y cliente para comunicarse. Puede ser un entero o un string con el número de puerto o un string con el nombre de un servicio (ej. 'http')

Comunicación con sockets

Secuencia de una comunicación con sockets TCP



1. El servidor ejecuta `socket()` para instanciar un objeto Socket (un socket servidor)
2. Usando este objeto ejecuta su método `bind(hostname, port)` para especificar dónde va a estar esperando conexiones de clientes
3. Luego el método `listen()` inicia el socket servidor que creamos (Empieza a "escuchar")
4. Cuando el cliente llama a `connect()` intenta establecer una conexión con el servidor. Se produce un handshake entre cliente y servidor para verificar que ambos pueden comunicarse.

Si este handshake es exitoso, el server completa la conexión con `accept()`

5. Eventualmente, se produce intercambio de datos con `recv()` y `send()` (para TCP) o `sendto()` y `recvfrom()` (UDP) - el servidor usa otro socket diferente al socket que se usó para establecer la comunicación...
6. Cuando el cliente cierra su conexión el servidor recibe `FIN` y cierra también su conexión

Ejemplo de servidor y cliente

- Un servidor TCP simple - devuelve lo que recibe

```
#!/usr/bin/env python3
import socket

HOST = '192.168.2.125' # IP del server
PORT = 5000            # port

# instancia socket - protocolos internet / TCP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# asocia el socket server con un host y puerto
s.bind((HOST, PORT))

# empieza a escuchar
s.listen()

# queda bloqueado hasta que recibe conexión -
# devuelve un socket (cliente) para comunicarse
# y la dirección y puerto del cliente (en IPv6 tiene dos elementos más)
conn, addr = s.accept()

print('Conectado con ', addr)

while True:
    # bloquea hasta recibir datos - máx 1024 bytes x vez
    data = conn.recv(1024)
    if not data: # si recibe b'' cliente cortó conexión
        break

    # devuelve todos los bytes en data
    conn.sendall(data)

conn.close()
s.close()
```

- Un ejemplo cliente para conectarse al servidor anterior:

```
#!/usr/bin/env python3
import socket

HOST = '192.168.2.125' # IP del server
PORT = 5000            # port del server
```

```
# instancia socket - TCP / IP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# conecta al servidor
s.connect((HOST, PORT))

while True:
    msg = input("mensaje a enviar: ")

    if msg == "x":
        break

    # envía datos
    s.sendall(bytes(msg, "utf-8"))

    # bloquea hasta recibir datos
    data = s.recv(1024)

    # muestra datos recibidos
    print('Respuesta: ', repr(data))

s.close()
```