

## ► Programación de Sockets

Normalmente cuando escribimos un programa, no necesitamos comunicarnos con otros programas o computadoras. Sin embargo es posible que necesitemos comunicarnos con otras computadoras, para crear aplicaciones de mensajería o otras aplicaciones con arquitectura servidor-cliente. Para crear tales aplicaciones, podemos usar la programación de sockets.

Se verán los conceptos básicos de programación de sockets y también implementaciones por separado aplicaciones de mensajería simple utilizando la programación de sockets TCP y UDP.

### ► ¿Qué Son los Sockets?

Cuando dos procesos o aplicaciones interactúan utilizando un canal específico, los sockets son los puntos finales o los puntos de entrada de dichos canales de comunicación.

Podemos usar sockets para establecer un canal de comunicación entre dos procesos, entre un proceso o en diferente proceso en diferentes máquinas. Hay diferentes tipos de sockets como socket TCP, socket UDP y socket de dominio UNIX.

### ► Cómo Implementar la Programación de Sockets.

Python nos proporciona un Módulo Socket para la programación de los sockets, el módulo socket es estándar en la biblioteca de Python y proporciona las métodos con la ayuda de los cuales puede crear sockets.

No se necesita descargar específicamente módulo socket en su máquina y puede importarlo directamente a su programa usando la declaración de importación de la sig. manera:

```
"import socket".
```

Para implementar la programación de socket necesitamos crear dos procesos que se comunican utilizando sockets uno de los programas estructura como servidor y el otro funcionaría como cliente. Tanto el servidor como el cliente tiene diferentes funcionalidades, por lo tanto usaremos diferentes funciones al crear procesos de servidor y cliente.

## ► Cómo Crear un Servidor en la Progr. de Sockets

Para crear un Servidor, primero creamos un Socket para ello utilizaremos el Método "socket"

### • Cómo Crear un Socket

(Socket()) \* La sintaxis es la siguiente:

```
import socket
```

Socket.socket(family = socket.AF\_INET, type = socket.SOCK\_STREAM,  
proto = 0, fileno = None)

- family: Representa la familia de direcciones a la que pertenece el Socket por defecto AF\_INET y crea un socket con una dirección de protocolo de Internet, versión 4.0 (IPv4) puede utilizarse la familia de direcciones AF\_UNIX para dirección UNIX y la AF\_INET6 (IPv6).
- type: Denota el tipo de socket y por defecto tiene SOCK\_STREAM indicando que el socket seguirá el protocolo TCP. Puede utilizar el SOCK\_DGRAM para protocolo UDP y crear sockets de datagramas que sigan el protocolo inferior.
- Proto: Denota el número del protocolo y normalmente es 0 y si se utiliza la familia de direcciones AF\_CAN en la familia de parámetros, el número de protocolos debe ser CAN\_RAW, CAN\_PCAN, CAN\_ISOTP, CAN\_J1939.

• fileno: Contiene el valor por defecto "None". Si especificamos un descriptor de fichero de fichero los valores de las características family, type y proto se detectan automáticamente a partir del descriptor de fichero.

Después de crear un socket lo vinculamos a una dirección y un número de puerto llamando "bind()".

## ► Unir el Socket a una dirección con el Método Bind()

- Usando la función Socket, el método bind() se invoca en el objeto socket que creamos, tiene una tupla que contiene la dirección a la que se unirá el socket el formato de la dirección puede variar según la familia de direcciones. Crearán con → por lo tanto la dirección consta de el nombre del host + el número del puerto

La sintaxis es la siguiente: bind(hostname, port) Puedes especificar el nombre del host específicamente si crea el servidor en la máquina local, puedes especificar el nombre del host "Local host." o como "127.0.0.1".

- Alternativamente puedes usar el método gethostname() para obtener el nombre del host. Puedes utilizar cualquier número del puerto superior (1024 - 65535)

Después de unir el socket a una dirección el servidor escucha las solicitudes de conexión del cliente y para ello creamos el método:

## listen()

### ► Escuchar las Conexiones "Listen()"

La sintaxis es: listen(backlog)

Aquí el parámetro "backlog" indica el número máximo de conexiones no aceptadas que el sistema permitirá antes de rechazar nuevas conexiones.

Después de listen, el servidor está listo para aceptar conexiones

## Aceptar una solicitud de conexión

El servidor se ejecuta en un bucle infinito y escucha las solicitudes de clientes para aceptar la conexión. Una vez que se encuentra una solicitud de cliente, el servidor acepta la solicitud utilizando el método `accept()`.

El método `accept` devuelve una tupla de direcciones (cliente, dirección). El cliente representa un nuevo objeto `socket` que usaremos para enviar y recibir mensajes, dirección es donde está vinculado.

## Comunicación con el cliente (`send()` y `recv()`)

Después de aceptar la conexión, el servidor puede comunicarse con el cliente usando el método `send` para enviar un mensaje al cliente. El método `send` se invoca sobre el cliente devuelto por el mensaje de aceptar y usamos el método `recv` para recibir los mensajes. El método `recv` cuando se invoca el cliente, acepta un número que representa el número máximo de bits que pueden leer la conexión. después de la ejecución se devuelven los datos leídos de la conexión. Una vez cumplimentadas todas las operaciones, debemos cerrar la conexión. Para ello invocamos el método `close()` sobre el objeto `client` devuelto por el método `accept()`. Despues de crear, ver todos los métodos para el servidor.

Ahora creamos un proceso del cliente que se comunica:

```
1 import socket
2 # Dirección y puerto del servidor
3 HOST= '127.0.0.1' # Dirección local (localhost)
4 PORT= 65432         # Puerto a escuchar
5 # Crear socket TCP/IP
6 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
7     s.bind((HOST, PORT)) # Vincular el socket a dirección y puerto
8     s.listen()           # Escuchar conexiones entrantes
9     print()
10    print('Servidor esperando...')
11    # Aceptar conexión
12    conn, addr = s.accept()
13    with conn:
14        print()
15        print('Servidor conectado a', addr)
16        while True:
17            data = conn.recv(1024)
18            if not data:
19                break
20            print()
21            print('El Servidor recibe', repr(data))
22            conn.sendall('Hola desde el servidor')
```

Ahora creamos un proceso de cliente que se comunica con el servidor

```
1 import socket
2
3 # Dirección y puerto del servidor
4 HOST = '127.0.0.1' # Dirección local (localhost)
5 PORT = 65432 # Puerto al que conectarse
6
7 # Crear un socket TCP/IP
8 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
9     s.connect((HOST, PORT)) # Conectarse al servidor
10    mensaje = "Hola desde el cliente!"
11    s.sendall(mensaje.encode()) # Enviar un mensaje al servidor
12    data = s.recv(1024) # Recibir la respuesta del servidor
13
14 print()
15 print(f'cliente recibe: {data.decode()}')
```

## COMO CREAR UN CLIENTE EN PROGRAMACIÓN SOC...

Para crear un cliente primero necesitamos crear un socket con el método socket como lo hicimos al crear el servidor. Recordar que los protocolos definidos para el socket del cliente deben ser los mismos que los del socket del servidor, de lo contrario el programa no funcionara correctamente.

Después de crear el socket necesitamos conectar el servidor usando `connect()`

### ► Conectarse al servidor (Connect) - 1 - (-1)

Sintaxis del método:

```
connect((Host, Port))
```

que proporciona el método bin al crear el servidor.

Aquí el parámetro Host denota la dirección del servidor. El parámetro port indica el número de puerto en el que se crea el socket del servidor. Debe dar los mismos valores como entrada al Host y el parámetro del puerto

### ► Comunicación con el servidor

Después de conectarse al servidor, puede comunicarse con el servidor utilizando los métodos `send()` y `recv()`. Finalmente es necesario cerrar la conexión del lado del cliente utilizando `close()`.

Para crear un proceso de cliente que siga el protocolo UDP, necesitamos crear un socket especificando **SOCK\_DGRAM** en la entrada de parámetros de tipo mientras creamos el socket del servidor con el método **socket**, no necesitamos usar el método **connect()** ya que aquí no requerimos crear una conexión.

Después de crear el socket, podemos comunicar a comunicarnos directamente con el servidor usando los métodos **sendto()** y **recvfrom()**. Después de comunicarte con el servidor no olvides cerrar el socket usando el método **close()** ~~exit~~

```
import socket

mySocket = socket.socket(
    family=socket.AF_INET, type=socket.SOCK_DGRAM, proto=0, fileno=None
)
print("Soket Creado")
while True:
    msg = "Hola, soy un cliente UDP creado por UPP"
    print("Envio de Mensaje al servidor:", msg)
    mySocket.sendto(msg.encode(), ("localhost", 9999))
    msg_in = mySocket.recv(1024).decode()
    print("Acuse de recibo recibido del servidor:")
    print(msg_in)
    print("Terminada la Conexión.")
    mySocket.close()
    break
```

## Programación de Sockets con el Protocolo UDP

Para crear una conexión con el protocolo UDP, debemos seguir los siguientes pasos mientras implementamos el servidor.

→ Especifique `Sock_DGRAM` en la entrada paramétrica de tipo al crear el socket del servidor en el método `socket()`.

→ Víncule el socket a una dirección y un número de puerto, utilizando el método `bind()`.

→ Cuando no necesitamos establecer una conexión con el cliente, no usamos los métodos `listen()` y `accept()`, para establecer la conexión. Directamente podemos empezar a comunicarnos con el cliente.

→ Para recibir un mensaje en el protocolo UDP, usamos el método `recvfrom()`, toma el número de bytes para leer como argumento de entrada y devuelve una tupla que contiene los datos y la dirección desde la que se ha recibido los datos.

→ Para enviar un mensaje en el protocolo UDP, usamos el método `sendto()` toma los datos como su primer argumento de entrada y una tupla que contiene el número del host y el número del puerto en la dirección del socket al que se envían los datos.

→ Después de la comunicación debemos cerrar el socket utilizando el método `close()`

```
import socket

mySocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM, proto=0, fileno=None)

print("Socket Creado")
direccion = "localhost"
puerto= 9999

mySocket.bind((direccion, puerto))
print("Socket enlazado a la dirección {} y el número de puerto{}".format(direccion, puerto))

while True:
    msg, client_addr= mySocket.recvfrom(1024)

    print("Mensaje recibido del cliente")
    print(msg.decode)
    print("Envío acuse de recibo al cliente")
    msg_out = "Mensaje recibido {}. Gracias.".format(msg).encode()
    mySocket.sendto(msg_out, client_addr)
    mySocket.close()
    break
```