

Python - Programação de soquete

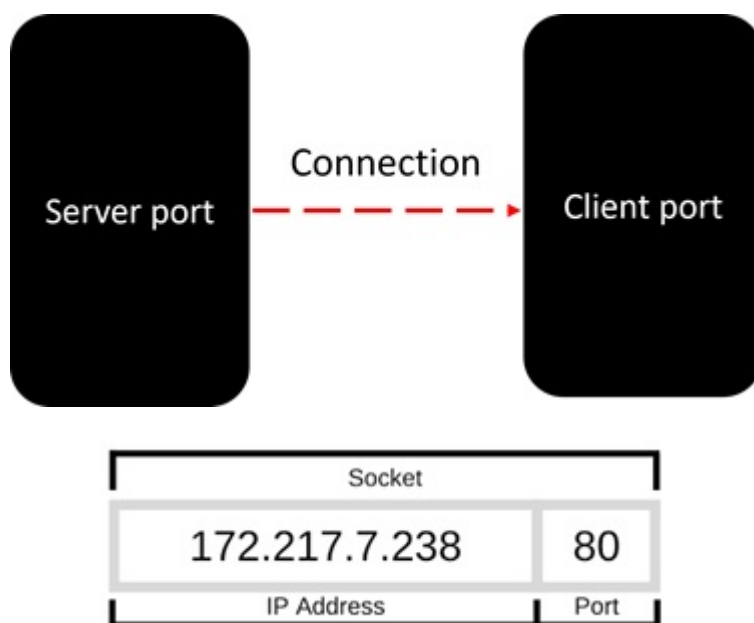
O módulo `socket` na biblioteca padrão incluía funcionalidades necessárias para comunicação entre servidor e cliente em nível de hardware.

Este módulo fornece acesso à interface do `socket` BSD. Está disponível em todos os sistemas operacionais, como Linux, Windows, MacOS.

O que são soquetes?

Sockets são os pontos finais de um canal de comunicação bidirecional. Os sockets podem se comunicar dentro de um processo, entre processos na mesma máquina ou entre processos em continentes diferentes.

Um socket é identificado pela combinação do endereço IP e do número da porta. Deve ser configurado corretamente em ambas as extremidades para iniciar a comunicação.



Os sockets podem ser implementados em vários tipos de canais diferentes: sockets de domínio Unix, TCP, UDP e assim por diante. A biblioteca de sockets fornece classes específicas para lidar com os transportes comuns, bem como uma interface genérica para lidar com o resto.

O termo programação de socket implica configurar sockets programaticamente para poder enviar e receber dados.

Existem dois tipos de protocolos de comunicação -



- protocolo orientado a conexão
- protocolo sem conexão

TCP ou Transmission Control Protocol é um protocolo orientado a conexão. Os dados são transmitidos em pacotes pelo servidor e montados na mesma ordem de transmissão pelo receptor. Como os soquetes em cada extremidade da comunicação precisam ser configurados antes de iniciar, esse método é mais confiável.

UDP ou User Datagram Protocol não tem conexão. O método não é confiável porque os soquetes não requerem o estabelecimento de nenhum processo de conexão e encerramento para a transferência dos dados.

Python O módulo de soquete

Este módulo inclui a classe Socket. Um objeto soquete representa o psir do nome do host e do número da porta. O método construtor tem a seguinte assinatura -

Sintaxe

```
socket.socket (socket_family, socket_type, protocol=0)
```

Parâmetros

- **family** - AF_INET por padrão. Outros valores - AF_INET6 (oito grupos de quatro dígitos hexadecimais), AF_UNIX, AF_CAN (Controller Area Network) ou AF_RDS (Reliable Datagram Sockets).
- **socket_type** - deve ser SOCK_STREAM (o padrão), SOCK_DGRAM, SOCK_RAW ou talvez uma das outras constantes SOCK_.
- **protocolo** - o número geralmente é zero e pode ser omitido.

Tipo de retorno

Este método retorna um objeto soquete.

Depois de ter o objeto soquete, você poderá usar os métodos necessários para criar seu programa cliente ou servidor.

Métodos de soquete de servidor

O soquete instanciado no servidor é chamado de soquete do servidor. Os métodos a seguir estão disponíveis para o objeto soquete no servidor -

- **Método bind()** - Este método vincula o soquete ao endereço IP e número da porta especificados.
- **Método listen()** - Este método inicia o servidor e executa um loop de escuta procurando a solicitação de conexão do cliente.
- **método accept()** - Quando a solicitação de conexão é interceptada pelo servidor, este método a aceita e identifica o soquete do cliente com seu endereço.

Para criar um soquete em um servidor, use o seguinte trecho -

```
import socket
server = socket.socket()
server.bind(('localhost',12345))
server.listen()
client, addr = server.accept()
print ("connection request from: " + str(addr))
```

Por padrão, o servidor está vinculado ao endereço IP da máquina local 'localhost', ouvindo um número de porta vazio arbitrário.

Métodos de soquete de cliente

Soquete semelhante é configurado no lado do cliente. Ele envia principalmente solicitações de conexão para o soquete do servidor que escuta seu endereço IP e número de porta

método conectar()

Este método usa um objeto tupla de dois itens como argumento. Os dois itens são o endereço IP e o número da porta do servidor.

```
obj=socket.socket()
obj.connect((host,port))
```

Assim que a conexão for aceita pelo servidor, ambos os objetos soquete podem enviar e/ou receber dados.

método enviar()

O servidor envia dados ao cliente usando o endereço que interceptou.

```
client.send(bytes)
```

O soquete do cliente envia dados para o soquete com o qual estabeleceu conexão.

método sendall()

semelhante a enviar(). No entanto, ao contrário de send(), este método continua a enviar dados de bytes até que todos os dados tenham sido enviados ou ocorra um erro. Nenhum é retornado em caso de sucesso.

método sendto()

Este método deve ser usado apenas no caso do protocolo UDP.

método recv()

Este método é usado para recuperar dados enviados ao cliente. No caso de servidor, utiliza o soquete remoto cuja solicitação foi aceita.

```
client.recv(bytes)
```

método recvfrom()

Este método é usado no caso do protocolo UDP.

DE ANÚNCIOS



Cruit

a new way to hire talent

Python - servidor de soquete

Para escrever servidores de Internet, usamos a função socket disponível no módulo socket para criar um objeto socket. Um objeto soquete é então usado para chamar outras funções para configurar um servidor de soquete.

Agora chame a função bind(hostname, port) para especificar uma porta para o seu serviço no host fornecido.

Em seguida, chame o método accept do objeto retornado. Este método espera até que um cliente se conecte à porta especificada e, em seguida, retorna um objeto de conexão que representa a conexão com esse cliente.

```
import socket
host = "127.0.0.1"
port = 5001
server = socket.socket()
server.bind((host,port))
server.listen()
```



```

conn, addr = server.accept()
print ("Connection from: " + str(addr))
while True:
    data = conn.recv(1024).decode()
    if not data:
        break
    data = str(data).upper()
    print (" from client: " + str(data))
    data = input("type message: ")
    conn.send(data.encode())
conn.close()

```

DE ANÚNCIOS

Python - cliente de soquete

Vamos escrever um programa cliente muito simples, que abre uma conexão para uma determinada porta 5001 e um determinado localhost. É muito simples criar um cliente de soquete usando a função de módulo de soquete do Python.

O **socket.connect(hostname, port)** abre uma conexão TCP com o nome do host na porta. Depois de abrir um soquete, você poderá lê-lo como qualquer objeto IO. Quando terminar, lembre-se de fechá-lo, como fecharia um arquivo.

O código a seguir é um cliente muito simples que se conecta a um determinado host e porta, lê todos os dados disponíveis no soquete e sai quando 'q' é inserido.

```

import socket
host = '127.0.0.1'
port = 5001
obj = socket.socket()
obj.connect((host,port))
message = input("type message: ")
while message != 'q':
    obj.send(message.encode())
    data = obj.recv(1024).decode()
    print ('Received from server: ' + data)
    message = input("type message: ")
obj.close()

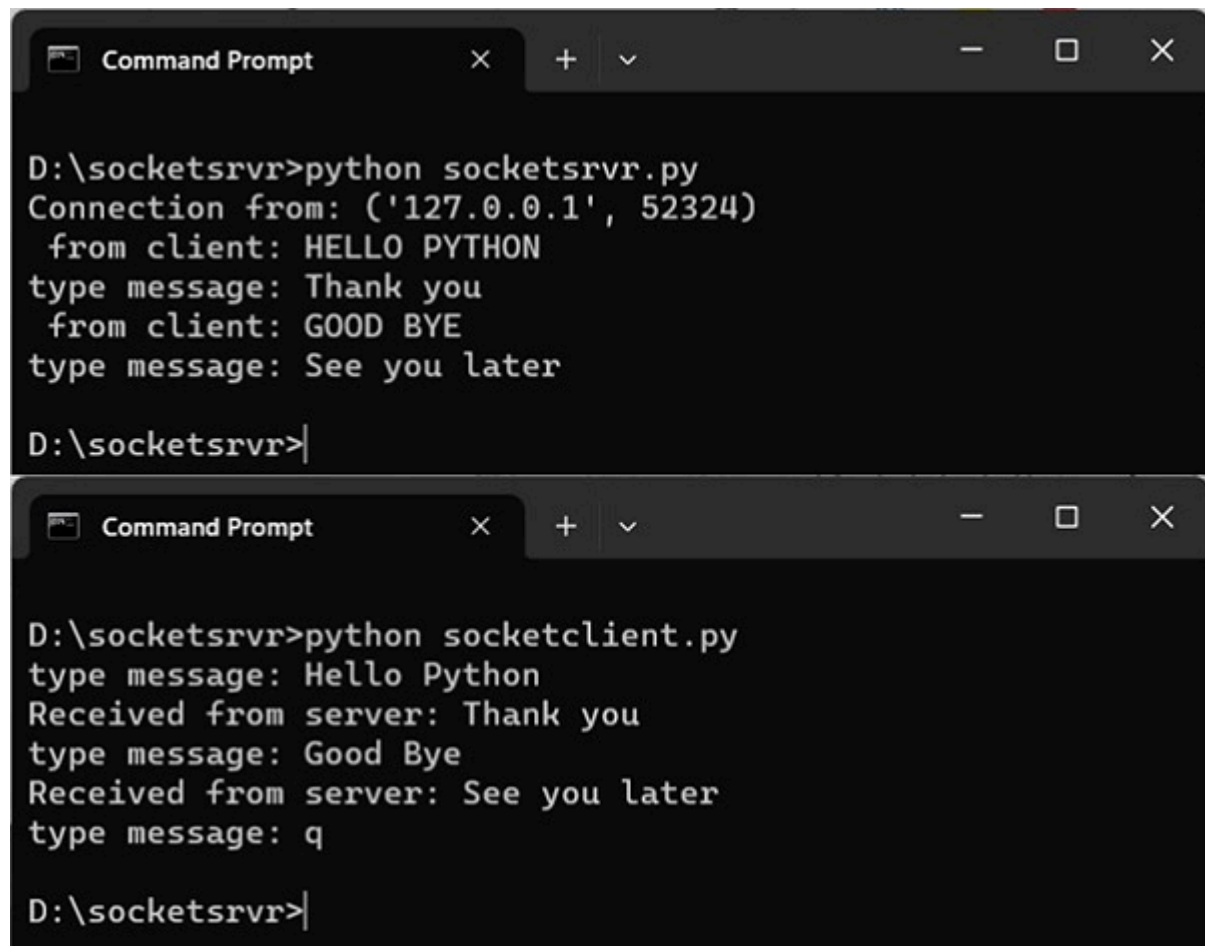
```

- Execute o código do servidor primeiro. Começa a ouvir.



- Em seguida, inicie o código do cliente. Ele envia solicitação.
- Pedido aceito. Endereço do cliente identificado
- Digite algum texto e pressione Enter.
- Os dados recebidos são impressos. Envie dados ao cliente.
- Os dados do servidor são recebidos.
- O loop termina quando 'q' é inserido.

A interação servidor-cliente é mostrada abaixo -



```
D:\socketsrvr>python socketsrvr.py
Connection from: ('127.0.0.1', 52324)
  from client: HELLO PYTHON
type message: Thank you
  from client: GOOD BYE
type message: See you later

D:\socketsrvr>

D:\socketsrvr>python socketclient.py
type message: Hello Python
Received from server: Thank you
type message: Good Bye
Received from server: See you later
type message: q

D:\socketsrvr>
```

Implementamos a comunicação cliente-servidor com módulo soquete na máquina local. Para colocar códigos de servidor e cliente em duas máquinas diferentes em uma rede, precisamos encontrar o endereço IP da máquina servidora.

No Windows, você pode encontrar o endereço IP executando o comando ipconfig. O comando ifconfig é o comando equivalente no Ubuntu.

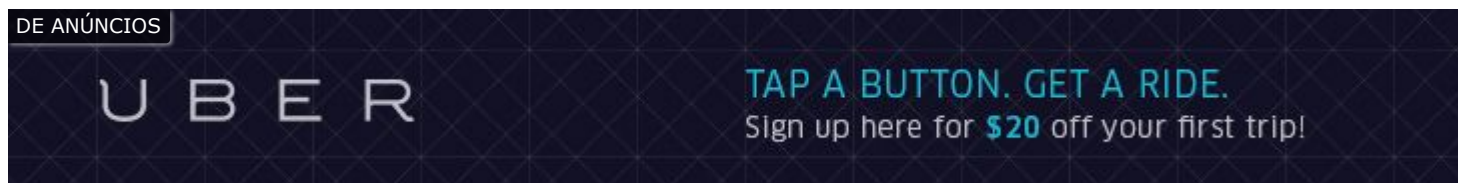
```
Command Prompt

Wireless LAN adapter Local Area Connection* 68:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . : 
Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix  . : ib-wrb304n.setup.in
Link-local IPv6 Address . . . . . : fe80::a1a2:ef95:61fa:9b7a%16
IPv4 Address. . . . . : 192.168.1.218
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
```

Altere a string do host nos códigos do servidor e do cliente com o valor do endereço IPv4 e execute-os como antes.



Transferência de arquivos Python com módulo Socket

O programa a seguir demonstra como a comunicação por soquete pode ser usada para transferir um arquivo do servidor para o cliente

Código do servidor

O código para estabelecer a conexão é o mesmo de antes. Após a solicitação de conexão ser aceita, um arquivo no servidor é aberto em modo binário para leitura, e os bytes são sucessivamente lidos e enviados ao fluxo do cliente até que o final do arquivo seja alcançado.

```
import socket
host = "127.0.0.1"
port = 5001
server = socket.socket()
server.bind((host, port))
server.listen()
conn, addr = server.accept()
data = conn.recv(1024).decode()
filename='test.txt'
f = open(filename,'rb')
while True:
    l = f.read(1024)
    if not l:
        break
    conn.send(l)
    print('Sent ',repr(l))
f.close()
```



```
print('File transferred')
conn.close()
```

Código do cliente

No lado do cliente, um novo arquivo é aberto no modo **wb** . O fluxo de dados recebidos do servidor é gravado no arquivo. Quando o fluxo termina, o arquivo de saída é fechado. Um novo arquivo será criado na máquina cliente.

```
import socket

s = socket.socket()
host = "127.0.0.1"
port = 5001

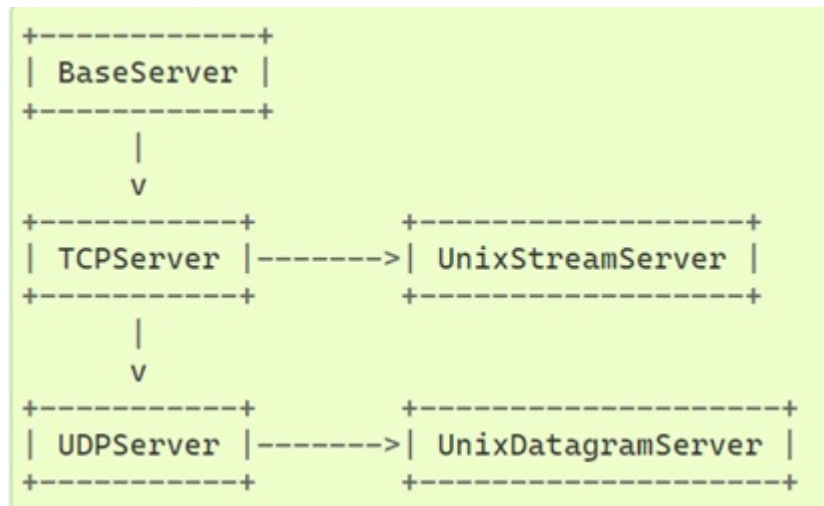
s.connect((host, port))
s.send("Hello server!".encode())

with open('recv.txt', 'wb') as f:
    while True:
        print('receiving data...')
        data = s.recv(1024)
        if not data:
            break
        f.write(data)

f.close()
print('Successfully received')
s.close()
print('connection closed')
```

Python O módulo socketserver

O módulo socketserver na biblioteca padrão do Python é uma estrutura para simplificar a tarefa de escrever servidores de rede. Existem as seguintes classes no módulo, que representam servidores síncronos -



Essas classes funcionam com classes RequestHandler correspondentes para implementar o serviço. BaseServer é a superclasse de todos os objetos Server no módulo.

A classe TCPServer usa o protocolo TCP da Internet para fornecer fluxos contínuos de dados entre o cliente e o servidor. O construtor tenta invocar automaticamente server_bind() e server_activate(). Os demais parâmetros são passados para a classe base BaseServer.

Você também deve criar uma subclasse da classe **StreamRequestHandler**. A TI fornece atributos self.rfile e self.wfile para leitura ou gravação para obter os dados da solicitação ou retornar dados ao cliente.

- **UDPServer** and **DatagramRequestHandler** - Essas classes devem ser usadas para o protocolo UDP.
- **DatagramRequestHandler** and **UnixDatagramServer** - Essas classes usam soquetes de domínio Unix; eles não estão disponíveis em plataformas não-Unix.

Código do servidor

Você deve escrever uma classe RequestHandler. Ele é instanciado uma vez por conexão com o servidor e deve substituir o método handle() para implementar a comunicação com o cliente.

```

import socketserver

class MyTCPHandler(socketserver.BaseRequestHandler):
    def handle(self):
        self.data = self.request.recv(1024).strip()
        host,port=self.client_address
        print("{}: {} wrote:".format(host,port))
        print(self.data.decode())
        msg=input("enter text .. ")
        self.request.sendall(msg.encode())
  
```

No número da porta atribuída ao servidor, um objeto da classe TCPServer chama o método forever() para colocar o servidor no modo de escuta e aceita solicitações recebidas de

clientes.

```
if __name__ == "__main__":
    HOST, PORT = "localhost", 9999
    with socketserver.TCPServer((HOST, PORT), MyTCPHandler) as server:
        server.serve_forever()
```

Código do cliente

Ao trabalhar com o socketserver, o código do cliente é mais ou menos semelhante ao do aplicativo cliente do soquete.

```
import socket
import sys

HOST, PORT = "localhost", 9999

while True:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        # Connect to server and send data
        sock.connect((HOST, PORT))
        data = input("enter text .. .")
        sock.sendall(bytes(data + "\n", "utf-8"))

        # Receive data from the server and shut down
        received = str(sock.recv(1024), "utf-8")
        print("Sent: {}".format(data))
        print("Received: {}".format(received))
```

Execute o código do servidor em um terminal de prompt de comando. Abra vários terminais para instâncias de clientes. Você pode simular uma comunicação simultânea entre o servidor e mais de um cliente.

Servidor	Cliente-1	Cliente-2
D:\socketsrvr>python meuservidor.py 127.0.0.1:54518 escreveu: do cliente-1 digite o texto.. olá 127.0.0.1:54522 escreveu: como vai digite o texto.. multar	D:\socketsrvr>python meuclient.py digite o texto .. . do cliente-1 Enviado: do cliente-1 Recebido: olá digite o texto .. . como vai Enviado:	D:\socketsrvr>python meuclient.py digite o texto .. . do cliente-2 Enviado: do cliente-2 Recebido: oi cliente-2 digite o texto .. . obrigado Enviado: obrigado

127.0.0.1:54523 escreveu:
do cliente-2
digite o texto..
oi cliente-2
127.0.0.1:54526 escreveu:
adeus
digite o texto..
Bye Bye
127.0.0.1:54530 escreveu:
obrigado
digite o texto..
tchau cliente-2

como vai
Recebido: bem
digite o texto .. .
adeus
Enviado: adeus
Recebido: tchau tchau
digite o texto .. .

Recebido:
tchau cliente-2
digite o texto .. .