Socket Programming

Background :

Sockets have a long history. Their use originated with ARPANET in 1971 and later became an API in the Berkeley Software Distribution (BSD) operating system released in 1983 called Berkeley sockets.

When the Internet took off in the 1990s with the World Wide Web, so did network programming. Web servers and browsers weren't the only applications taking advantage of newly connected networks and using sockets. Client-server applications of all types and sizes came into widespread use.

Today, although the underlying protocols used by the socket API have evolved over the years, and we've seen new ones, the low-level API has remained the same.

The most common type of socket applications are client-server applications, where one side acts as the server and waits for connections from clients. This is the type of application that I'll be covering in this tutorial. More specifically, we'll look at the socket API for Internet sockets, sometimes called Berkeley or BSD sockets. There are also Unix domain sockets, which can only be used to communicate between processes on the same host.

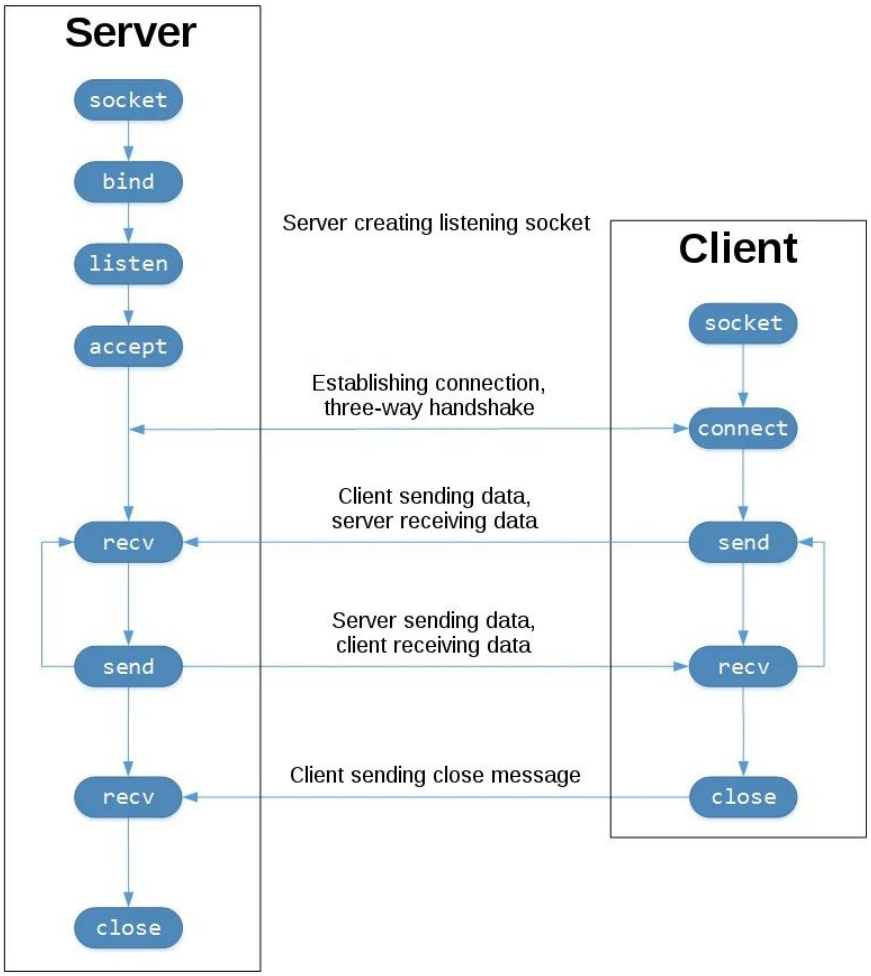
Socket API Overview

Python's socket module provides an interface to the Berkeley sockets API. This is the module that we'll use and discuss in this tutorial.

The primary socket API functions and methods in this module are:

- socket()
- bind()
- listen()
- accept()
- connect()
- connect_ex()
- send()
- recv()
- close()

Python provides a convenient and consistent API that maps directly to these system calls, their C counterparts.



The left-hand column represents the server. On the right-hand side is the client
Starting in the top left-hand column, note the API calls the server makes to setup a “listening” socket:

- socket()
- bind()
- listen()
- accept()

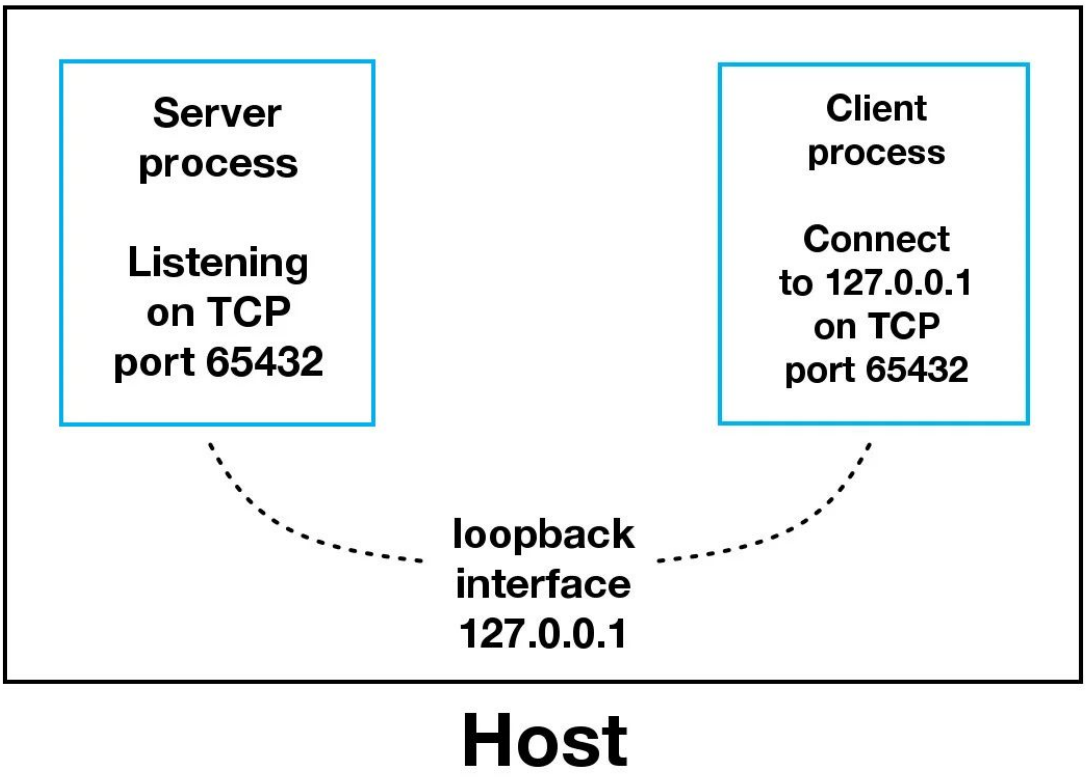
A listening socket does just what it sounds like. It listens for connections from clients. When a client connects, the server calls accept() to accept, or complete, the connection.

The client calls connect() to establish a connection to the server and initiate the three-way handshake. The handshake step is important since it ensures that each side of the connection is reachable in the network, in other words that the client can reach the server and vice-versa. It may be that only one host, client or server, can reach the other.

In the middle is the round-trip section, where data is exchanged between the client and server using calls to send() and recv().

At the bottom, the client and server close() their respective sockets.

Server-Client interaction:



Server.py ⇒

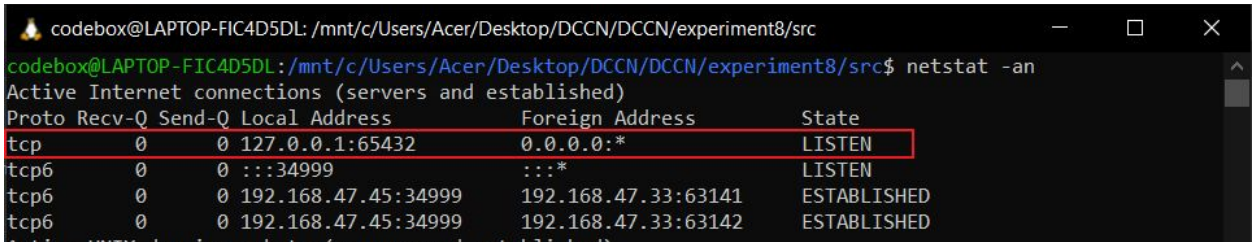
```python
import socket

HOST = '127.0.0.1'  # Localhost address
PORT = 65432        # Port to listen ( non sudo ports > 1023)

# using context manager type (we don't need to close the socket)
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    """
    socket.AF_INET - For IPv4
    socket.SOCK_STREAM - For TCP protocol
    """
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
```

```
            if not data:
                break
            conn.sendall(data)
```

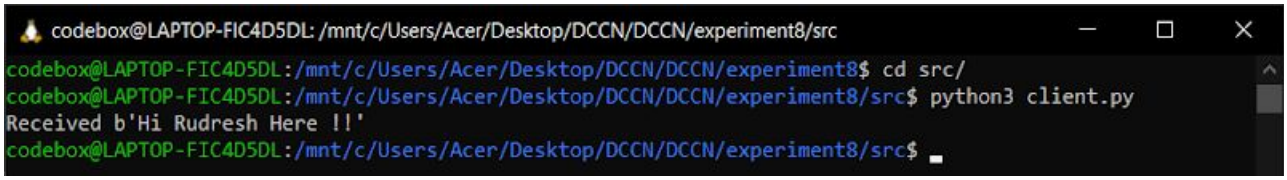
Client.py ⇒

```
import socket

HOST = '127.0.0.1'  # The server's hostname or IP address
PORT = 65432        # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hi Rudresh Here !!')
    data = s.recv(1024)

print('Received', repr(data))
```

Results ⇒
After starting script server.py we can see a new connection listening on port 65432



Then when we start client.py we'll see the output ⇒



Conclusion ⇒

I understood the basics of socket programming and established a simple echo connection between client and server using the same.

References ⇒

https://realpython.com/python-sockets/
https://www.geeksforgeeks.org/socket-programming-python/
https://docs.python.org/3/library/socket.html