



Programming and Communications III: Sockets

Jordi Ricard Onrubia Palacios

Departament d'Informàtica i Enginyeria Industrial Universitat de Lleida



Programming and Communications II: Overview



Overview



Standard Library

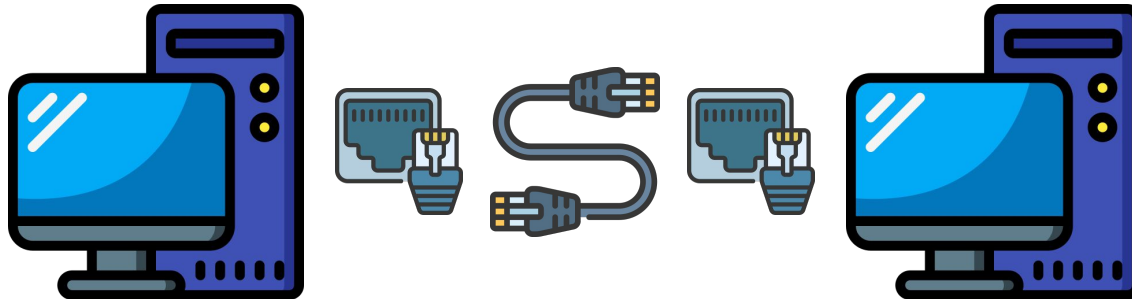


Sockets Code

Overview

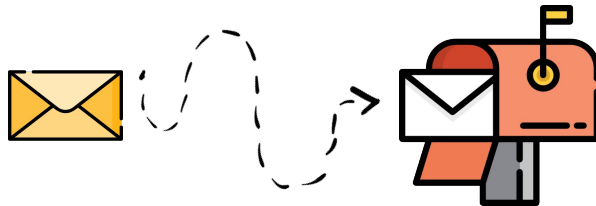
Sockets are communication mechanisms, making possible transfer information between different processes even in different machines.

Sockets are assigned to a port in a machine, so they can be identified and accessed by the TCP layer. (A port is an interface from where data can be sent and received)

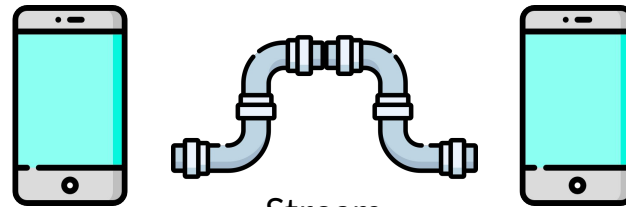


Types of Sockets

- **Datagram (UDP):** No-connection oriented socket, your data is sent to another socket, like sending letters by mail.
- **Stream (TCP):** Connection oriented socket, sequenced and a unique flow of data with mechanisms for creating and destroying connections and detecting errors. A connection is established between two sockets and data is transferred between them. Like calling by phone, a connection is established and a conversation is done until finished.



Datagram



Stream



Programming and Communications II: Standard Library



Overview

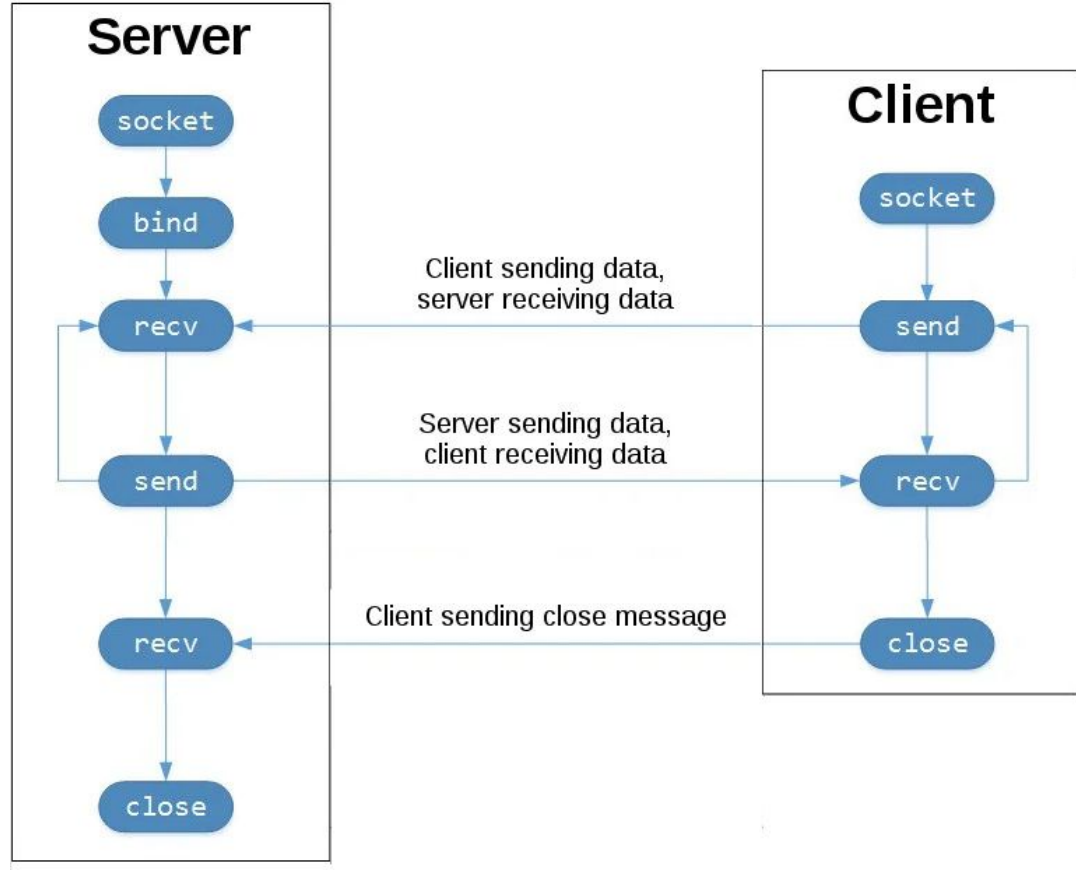


Standard Library

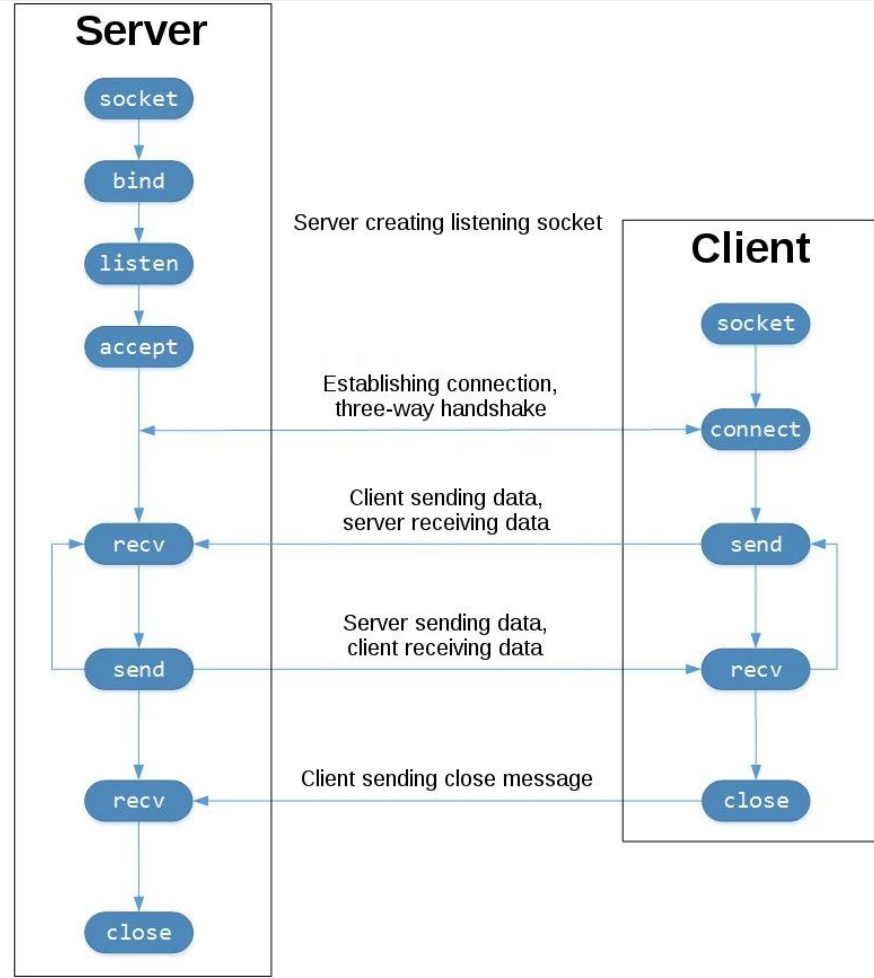


Sockets Code

Common Architectures: Client-Server UDP



Common Architectures: Client-Server TCP



Python Libraries: Standard Library



Socket Creation:

To create a socket we need to send to the constructor the family type of the socket, in our case we will use always the `socket.AF_INET`, and the second parameter the type of the socket, as we have seen before we have the datagram socket, referred in python as `socket.SOCK_DGRAM` and the streaming socket, referred as `socket.SOCK_STREAM`.



```
# Datagram socket type UDP
socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Datagram socket type TCP
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```


Python Libraries: Standard Library



Socket Server Preparation:

- Binding: If we are using UDP sockets we will need to bind one socket to a port, this is done by the server part. After this step, the UDP socket is ready to receive messages. For TCP sockets we must add one step to be ready to communicate.
- Listening: TCP sockets require to listen for requests to establish a conversation. Once the TCP server hears a connection request, it will be in charge of accepting the request to begin to communicate.
- Accept: Once accepted, we will receive the connection, that will be used for further communications and the address of the client socket.

Python Libraries: Standard Library



```
# Datagram socket type UDP, binding step  
s.bind((HOST, PORT))  
  
# Datagram socket type TCP, binding step  
s.bind((HOST, PORT))  
s.listen()  
conn, add = s.accept()
```

Python Libraries: Standard Library

Socket Client Connection:

- For UDP clients we do not need to do anything else, once the server is available to receive messages, the client only needs to send messages.
- For TCP clients we need to send to the TCP server a connection request (we have seen previously that the TCP server needs to accept the client request to establish a connection). Once the connection request is accepted, the client will be able to send messages.




```
# Streaming socket TCP Socket Connection  
s.connect((HOST, PORT))
```

Python Libraries: Standard Library



Socket Data Sending:

We have two different ways of sending data, if we are using a UDP socket, we are sending a message to an specific location, we do not have any connection established, therefore, we need to specify the receiver. When we are working with TCP sockets, we are establishing a connection, this connection is returned at the time of accepting the request from the client. Therefore we can use that connection to return a message. Otherwise, we can use the address also returned by the accept to send a message.



```
# Sending a message with the socket
s.sendto(data, (HOST, PORT))
# Sending a message with the connection returned
# by the accept
conn.sendall(data)
```

Python Libraries: Standard Library

Socket Data Receiving:

The same as before, we have different ways of receiving data, if we do not have a connection as in the UDP we can use the `socket.recvfrom`, this will return the data and the address from where this data is coming from, very useful for when we do not have a connection oriented. When we have a connection between the client and the server, we can use the `socket.recv`, this will only return the data. Note that both methods expect the number of bytes to be retrieved from the message. If we pass less bytes than the size of the data received, this data will be cut.

```
# Recvfrom returns a tuple (data, address)
data, address = s.recvfrom(1024)
# Recv returns only the data
data = s.recv(1024)
```

Python Libraries: Standard Library

Attention:

Check that every time we are communicating, we are doing it directly from the client to the client, therefore, if the client or the server losses the communication, the process of sending the message will throw an error. Therefore, the python standard socket library requires a persistent connection.





Programming and Communications II: Sockets Code



Overview




Standard Library



Sockets Code

Sockets Code: TCP Client



```
# TCP-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"Hello, world")
    data = s.recv(1024)

print(f"Received {data!r}")
```


Sockets Code: TCP Server



```
# TCP-server.py

import socket

HOST = "127.0.0.1" # Standard loopback interface address (localhost)
PORT = 65432 # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

Sockets Code: UDP Client

```
# UDP-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_DGRAM) as s:
    print(f"Sending message to server")
    s.sendto(b"Hello, world", (HOST, PORT))
    data = s.recvfrom(1024)
    print(f"Received {data}")
```

Sockets Code: UDP Server



```
● ● ●  
  
# UDP-server.py  
  
import socket  
  
HOST = "127.0.0.1" # Standard loopback interface address (localhost)  
PORT = 65432 # Port to listen on (non-privileged ports are > 1023)  
  
with socket.socket(socket.AF_INET, type=socket.SOCK_DGRAM) as s:  
    s.bind((HOST, PORT))  
    print(f"Server listening: {HOST} {PORT}")  
    while True:  
        data = s.recvfrom(1024)  
        print(f"Received from client: {data}")  
        if not data:  
            break  
        num_bits = s.sendto(data[0], data[1])  
        print(f"Sent to client: {data[0]} {num_bits}")
```

Sockets Code: Links



TCP Sockets: [Link](#)

UDP Sockets: [Link](#)