Les Sockets en Python

P.PISZYNA – Juin 2022

Généralités

- Le mot « Socket » désigne une prise permettant ici de raccorder une connexion à travers le réseau.
- Les sockets procurent une interface de communication inter-process (IPC) à travers le système d'exploitation.
- Pour mettre en œuvre une socket, il faudra :
  - Un point d'extrémité de connexion (adresses IP du client et du serveur),
  - Un process utilisant cette connexion (numéro de port).
- La programmation avec les sockets consiste à utiliser un ensemble de fonctions disponibles dans tous les langages.

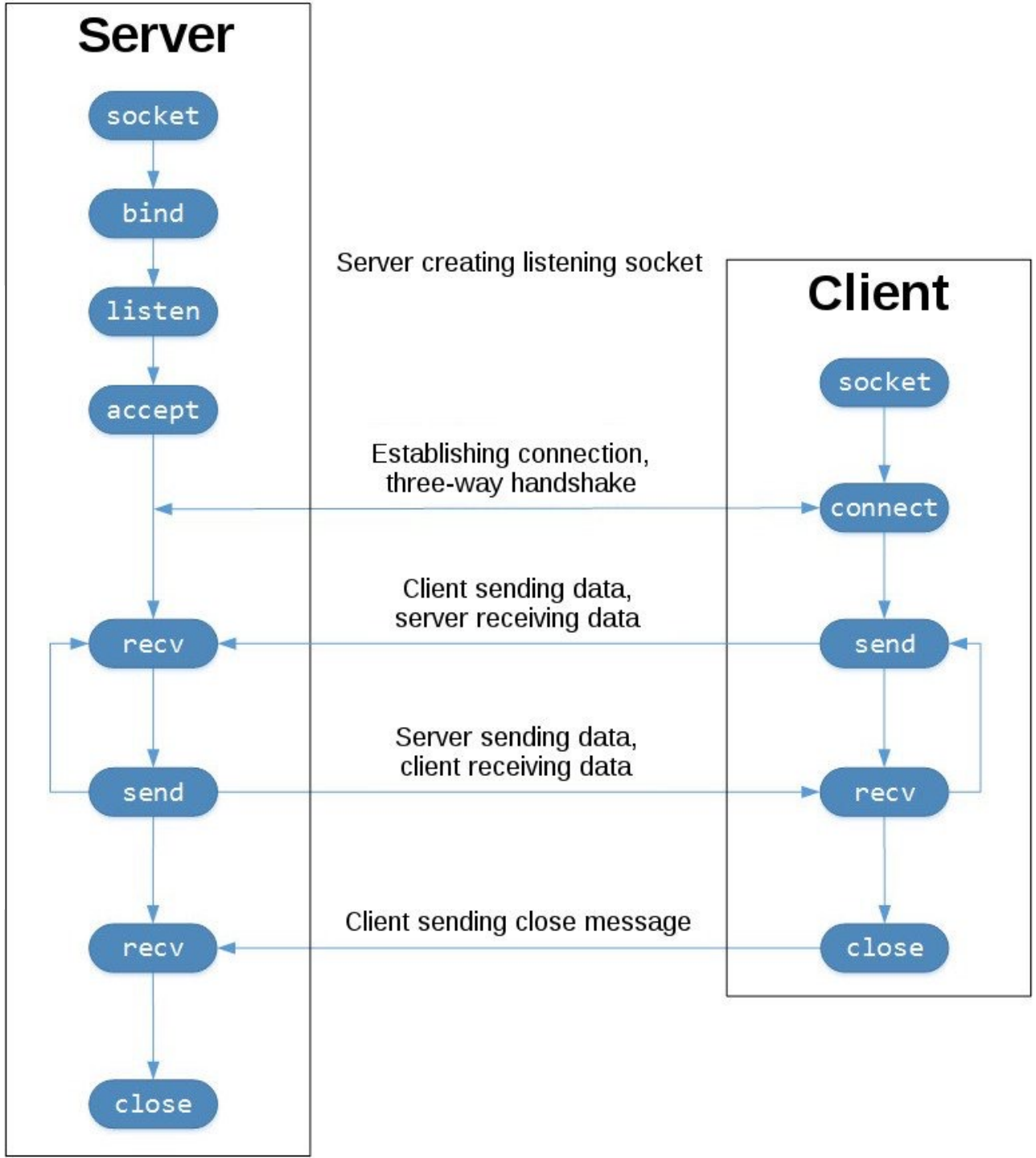
Mécanisme

Le *serveur* :

1. attend une connexion de la part du client ;
2. accepte la connexion quand le client se connecte ;
3. échange des informations avec le client ;
4. ferme la connexion.

Le *client* :

1. se connecte au serveur ;
2. échange des informations avec le serveur ;
3. ferme la connexion.



Fonctions de l’objet socket en Python

- **accept()** : accepte une connexion, retourne un nouveau socket et une adresse client
- **bind(addr)** : associe le socket à une adresse locale
- **close()** : ferme le socket
- **connect(addr)** : connecte le socket à une adresse distante
- **getpeername()** : retourne l'adresse distante
- **listen(n)** : commence à écouter les connexions entrantes
- **recv(buflen[, flags])** : recoit des données
- **recvfrom(buflen[, flags])** : reçoit des données et l'adresse de l'envoyeur
- **sendall(data[, flags])** : envoie toutes les données
- **send(data[, flags])** : envoie des données mais il se peut que pas toutes n’y soient
- **shutdown(how)** : fermer les connexions dans un ou les deux sens

Exemple 1 : Echo Server

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

HOST = "0.0.0.0"  # Any interface address (localhost)
PORT = 65432  # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
    client.bind((HOST, PORT))
    client.listen()
    connection, address = client.accept()
    print(f"Connected by {address}")
    with connection:
        while True:
            data = connection.recv(1024)
            if data == (b'\r\n'):
                break
            print(data)
            connection.sendall(data)
```

Quelques notions nouvelles en Python

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
    client.bind((HOST, PORT))
    client.listen()
    connection, address = client.accept()
```

Équivaut à

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try :
    client.bind((HOST, PORT))
    client.listen()
    connection, address = client.accept()
finally :
    client.close
```

Quelques notions nouvelles en Python

```
language = "Python"
school = "freeCodeCamp"
print(f"I'm learning {language} from {school}.")
```

```
#Output
I'm learning Python from freeCodeCamp.
```

ATTENTION

Les données échangées sont de type `bytes` !!

En Python, les **chaînes d'octets** (bytes) sont représentées comme des chaînes de caractères (limité au codage ascii), préfixées par un `b` : `b'Bonjour !'`

- Conversion `str` → `bytes` : `bytes("donnée", "utf-8")` → `b'donn\xc3\xa9e'`
- Conversion `bytes` → `str` : `b'donn\xc3\xa9e'.decode()` → `"donnée"`

Exemple 2 : Client

```
1   #!/usr/bin/env python3
2   import socket
3
4   HOST = "192.168.1.44"  # Server interface address
5   PORT = 65432  # Port to connect on (non-privileged ports are > 1023)
6
7   with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
8       server.connect((HOST, PORT))
9       server.sendall(b'Bonjour')
10      data = server.recv(1024)
11
12  print(f"Received {data!r}")
```

Exercice 1

Implémenter en Python un programme « Serveur » qui :

- créé un *socket* d'écoute et l'associe au port 63000 ,
- le met à l'état d'écoute et attend qu'un client s'y connecte
- affiche l'adresse du client qui vient de se connecter
- attend la réception d'un message de la part du client
- le renvoie aussitôt au client en rajoutant à la fin : « \nPrésent ! »
- ferme les connexions

Exercice 2

Implémenter en Python un programme « Client » qui :

- crée un *socket*
- demande une connexion au serveur d'adresse *('localhost', 63000)*
  *'localhost' est le nom d'hôte qui désigne l'**interface de bouclage** (loopback interface), c'est à dire la machine locale. Cela correspond à l'adresse IP '127.0.0.1')*
  Le client et le serveur sont sur la même machine ici !
- envoie un message « Serveur es-tu là ? »
- affiche la réponse du serveur
- ferme la connexion

Les threads en Python

```python
#!/usr/bin/env python3
import threading
from time import sleep

def fonction(n):
    for i in range(5):
        print(f"Thread {n}: on est dans la boucle {i}")
        sleep(0.5)

# création de thread
t1 = threading.Thread(target=fonction, args=(1,))
t2 = threading.Thread(target=fonction, args=(2,))

# démarrer le thread t1
t1.start()
# démarrer le thread t2
t2.start()

# attendre que t1 soit exécuté
t1.join()
# attendre que t2 soit exécuté
t2.join()

# les deux thread sont exécutés
print("C'est fini!")
```

Comment exécuter le thread 2 deux fois plus rapidement que le thread 1 ?

Création d’un serveur multi-connexions :

```
class ClientThread(threading.Thread):

    def __init__(self, ip, port, clientSocket):
        threading.Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.clientSocket = clientSocket
        print("[+] Nouveau thread pour %s %s" % (self.ip, self.port, ))

    def run(self):
        print("Connexion de %s %s" % (self.ip, self.port, ))
        data = self.clientSocket.recv(2048)
        print(data)
        self.clientSocket.close()
        print("Client déconnecté...")
```

Création d’un serveur multi-connexions :

```
listeningSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listeningSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listeningSocket.bind(("",65432))

while True:
    listeningSocket.listen(10)
    print( "En écoute...")
    (clientSocket, (ip, port)) = listeningSocket.accept()
    newThread = ClientThread(ip, port, clientSocket)
    newThread.start()
```

Exercice 3 :

Par groupe de 2 personnes, implémenter en Python deux programmes (« Client » et « Serveur ») qui communiquent, entre deux machines différentes, selon le protocole suivant :

Une fois la connexion établie, entre le *client* et le *serveur*, les deux utilisateurs peuvent échanger des messages :

- coté *client*, l'utilisateur est invité à saisir un message, puis ce message est envoyé au *serveur*
- coté *serveur*, l'utilisateur reçoit le message du *client*, puis est invité à son tour à répondre
- la discussion continue ainsi jusqu'à ce qu'un des participants envoie le message « fini »
- la connexion coté client est alors fermée et le serveur est remis dans l'état d'attente

Webographie des captures d'écran (du plus simple au plus compliqué) :

- https://info.blaisepascal.fr/nsi-sockets-python
- https://python.doctor/page-reseaux-sockets-python-port
- https://diu-eil.gricad-pages.univ-grenoble-alpes.fr/archi-robotique-systeme-reseau/reseaux/cours_sockets_python.pdf
- https://realpython.com/python-sockets/
- https://moodle1.u-bordeaux.fr/course/view.php?id=4713
- https://docs.python.org/3/library/socket.html (Référence)