Python - Révisions et introduction aux sockets

Un *socket* est un objet permettant d'ouvrir une connexion et de communiquer avec une machine locale ou distante. Il a été introduit dans les distributions de Berkeley (un système UNIX).

Dans le cas d'une communication client-serveur, il y aura deux programmes distincts. La documentation de Python propose deux programmes qu'on retrouvera ci-dessous [1].

```python
# Echo server program
import socket

HOST = ''                 # Symbolic name meaning all available interfaces
PORT = 50007              # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024) #we receive data by 1024 bytes chunks
            if not data: break
            print("received:", data)
            conn.sendall(data)
```

```python
# Echo client program
import socket

HOST = 'localhost'    # The remote host
PORT = 50007          # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    print("message sent to server")
    data = s.recv(1024) #we receive data by 1024 bytes chunks
    print('Received', repr(data))
```

Observations :

- On prendra soin d'utiliser un port non-réservé (> 1024) pour être sûr de ne pas bloquer les autres applications.
- Les méthodes `bind` et `recv` du socket sont **bloquantes** : elles mettent en pause le fil d'exécution jusqu'à ce qu'un certain événement se réalise (voir fig. 1).

1. https://docs.python.org/fr/3/library/socket.html#example

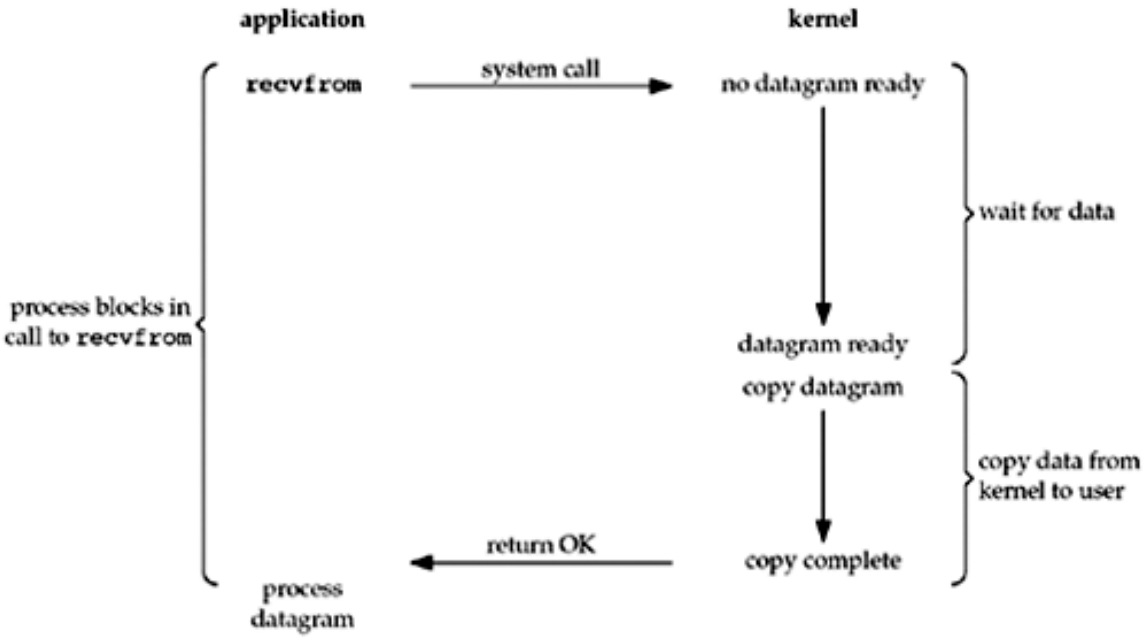

FIGURE 1 – Schéma d'un socket bloquant.

— Les données envoyées et reçues le sont sous forme d'octets. Il faut donc convertir la chaîne de caractères en liste de *bytes* (en la précédant de la lettre b) pour l'envoyer.

— Le socket est muni d'un *buffer* à partir duquel on lit les données avec la méthode `recv`, en indiquant le nombre d'octets qui seront lus (voir fig. 2).

Attention : envoyer des données avec `send` ou `sendall` ne garantit pas la réception totale de l'autre côté ! De même, l'émetteur et le récepteur doivent se mettre d'accord sur une syntaxe pour délimiter les messages. Si on envoie un texte de 8000 octets, et que le récepteur ne reçoit que 2500 octets, ce dernier ne peut pas savoir s'il a reçu l'intégralité.

Exercice 1 — Serveur multi-fonctions

En reprenant les codes présentés ci-dessus, rédiger deux programmes client-serveur où le client envoie des requêtes sous forme de chaîne de caractères au serveur, puis attend la réponse. Le serveur traite la requête puis envoie la réponse. Les programmes tournent tant que l'un des deux sockets n'est pas fermé.

Exemple de commande :

— `UPPER <str>` et `LOWER <str>` : retourne la chaîne où tous les caractères seront transformés en majuscules (resp. minuscules) ;

— `DATENOW` : retourne le timestamp actuel ;

Exercice 2 — Nombre mystère

Ce programme est généralement étudié au début de l'apprentissage de la programmation. On rappelle les règles : un nombre entier est généré aléatoirement entre deux bornes, et un joueur a un certain nombre d'essais pour le trouver. Pour chaque proposition du joueur, l'ordinateur doit donner une indication (plus ou moins).

Implémenter le jeu, en créant un programme serveur, qui génère un nombre, et un programme client, qui se connecte au serveur et joue au jeu. On effectuera une double

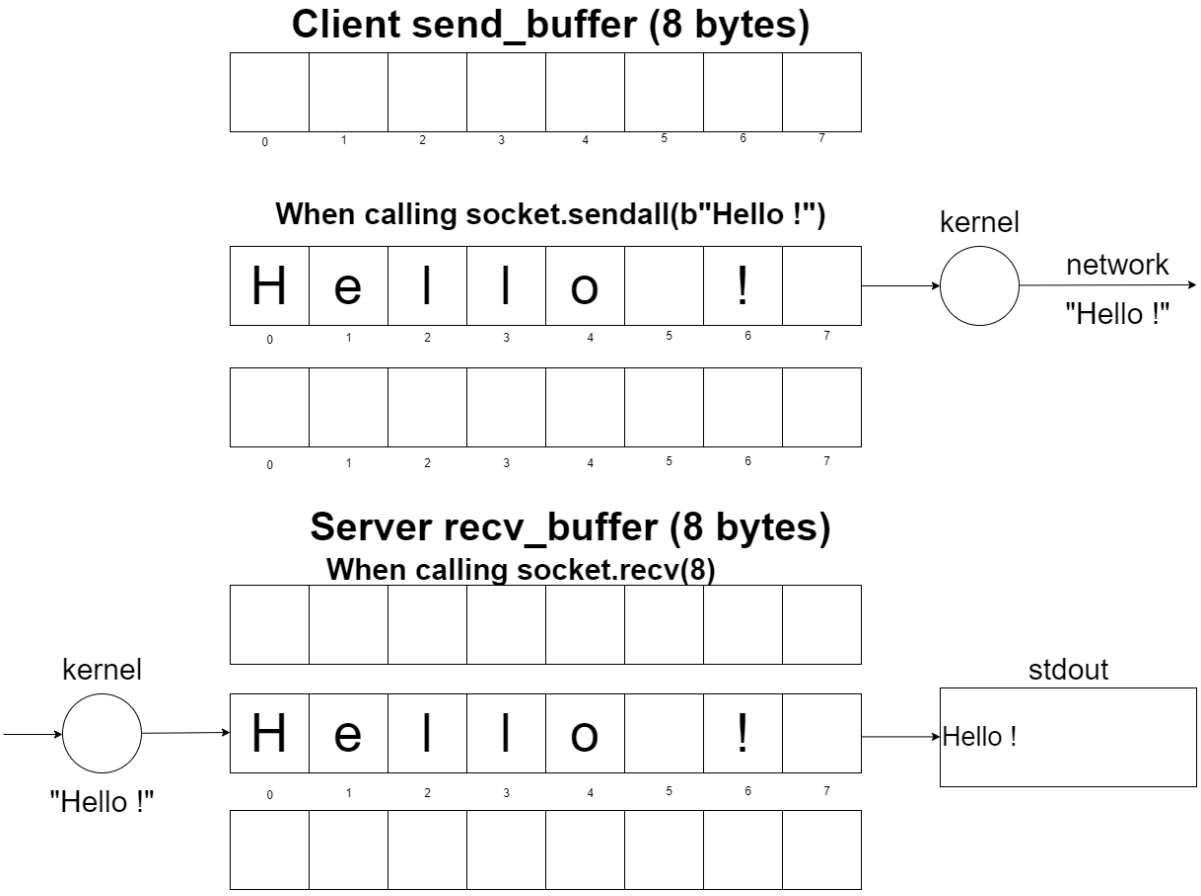

FIGURE 2 – Le fonctionnement derrière les coulisses des sockets de Python.

vérification : le client vérifie que la chaîne de caractères tapée au clavier est bien un nombre, et le serveur vérifie que la chaîne de caractères reçue est bien celle d'un nombre.

Exercice 3 — Morpion

Chaque joueur donne son nom, puis le serveur distribue les pièces (X ou O) aléatoirement à chaque joueur. Le joueur avec un X commence.

On représente le terrain avec une liste de listes de caractères. `"-"` représente une case vide.

On implémente les fonctions suivantes dans les deux programmes :

- `get_mark_by_pos` qui prend en paramètre le terrain une position du terrain et retourne le contenu de la case correspondante ;
- `get_line_from_field` qui prend en paramètre le terrain et une liste de positions et qui retourne la liste du contenu de ces cases ;
- `place_mark` qui prend en paramètre le numéro du joueur, le terrain et la position, et qui place la marque (donc modifie un élément de la liste) correspondant au joueur ;
- `is_ended` qui prend en paramètre le terrain et un numéro de joueur et retourne un booléen indiquant si une ligne a été réalisée par ce joueur ;
- `show_field` qui affiche le terrain de manière lisible ;
- `check_correct_input` qui prend en paramètre une chaîne de caractère et le terrain, et qui vérifie si la commande entrée est correcte. Une commande correcte est de la forme `"x y"`, avec x et y deux chiffres entre 0 et 3, séparés par un espace. On vérifie en outre que la case ne contienne pas déjà une marque. La fonction retourne un booléen.
- `input_str_to_position` qui prend en paramètre un string correspondant à un coup, et qui retourne les coordonnées du coup sous forme de tuple
- `input_position` qui prend en paramètre le terrain, et qui retourne une chaîne de caractère (entrée par l'utilisateur) qui correspond à un coup joué. La fonction doit impérativement retourner un coup valide et demande à l'utilisateur de recommencer tant que son entrée n'est pas valide.

  Logique de jeu :

  - Le serveur génère les numéros de joueurs (1 et 2) aléatoirement, et envoie au client son numéro. Le client attend donc au départ un entier.
  - Tant que la partie n'est pas finie :
  - Le serveur joue un coup, et attend la réponse du client **si ce coup ne provoque pas la victoire** ;
  - Le client attend le coup du serveur et joue **si ce coup ne provoque pas la victoire** ;

1 Dictionnaires

Un dictionnaire (tableau associatif) est une structure de données qui associe à un ensemble de clefs un ensemble de valeurs : chaque clef est associée à au plus une valeur. Par exemple, un annuaire téléphonique associe à chaque numéro de téléphone une personne.

Cependant, une telle structure n'est pas ordonnée, donc peu adaptée pour effectuer des tris.

On effectue en général les opérations suivantes :

— ajout et suppression d'éléments ;

— recherche ;

— modification.

Nous nous penchons ici sur le type `dict` de Python, mais les règles énoncées seront sensiblement identiques pour les autres langages.

Un dictionnaire se déclare avec des accolades. Comme les listes, il est possible de les déclarer par compréhension.

```
d1 = {"nom" : "Paul", "prenom": "Pina-Gherardi", "EID": 20140163}
d2 = {}
L = ["Paul", "Pierre",  "Jacques"]
d3 = {name: random.randint(15, 25) for name in L}
```

Trois méthodes permettent d'obtenir ses éléments sous forme de `view object`, qu'on utilisera comme itérateurs pour les boucles :

— `keys` retourne un `view object` des clefs du dictionnaire. Note : effectuer un cast en liste donne également la liste des clefs ;

— `values` retourne un `view object` des clefs du dictionnaire.

— `items` retourne un `view object` des couples (clef, valeur) du dictionnaire sous forme de tuples.

```
for k, v in dico.items():
    print(k,":",v)
```

On gardera en tête l'idée qu'il est tout à fait possible d'imbriquer des dictionnaires dans d'autres dictionnaires, ou de créer des listes de dictionnaires, etc.

Exercice 4 — Comptage

Créer une fonction prenant en paramètre une liste d'éléments immuables (nombres, chaînes, tuples, etc), et qui retourne un dictionnaire comptant le nombre d'occurrences de chaque élément.

Exercice 5 — Scrabble

Créer une fonction retournant un nombre de points correspondant au mot joué. La fonction prend en paramètre un dictionnaire (qu'on créera en tant que constante) qui associe à chaque lettre de l'alphabet un nombre de points. Elle prend également un dictionnaire contenant toutes les données nécessaires au calcul :

— le mot joué ;

— la liste des positions des lettres positionnées sur des cases bleu clair ;

— la liste des positions des lettres positionnées sur des cases bleu foncé ;

— la liste des positions des lettres positionnées sur des cases rose ;

— la liste des positions des lettres positionnées sur des cases rouges ;

On donne les effets des cases :

— bleu clair : multiplie la valeur de la lettre par 2

— bleu foncé : multiplie la valeur de la lettre par 3

— rose : multiplie la valeur du mot par 2

— rouge : multiplie la valeur du mot par 3

On applique d'abord les multiplicateurs des lettres puis ceux des mots.

2 JSON

Le JSON (**J**ava**S**cript **O**bject **N**otation) est un format de données textuelles et structurées. Le JSON est soutenu par de nombreuses bibliothèques dans les langages de programmations usuels, dont le Python. On l'utilise comme format commun pour communiquer des données complexes, par sérialisation (conversion de données en série de bits) dans le Web, et plus généralement dans les communications entre machines.

2.1 Types de données

JSON manipule peu de types de données. On compte deux types composés :

— des tableaux de valeurs ;

— des couples clef/valeur (objets, équivalents des dictionnaires en Python).

Ainsi que 4 types primitifs (scalaires) :

— booléens ;

— nombres ;

— chaînes de caractères ;

— la valeur `null`.

On présente ci-dessous un exemple relativement fourni d'un fichier JSON [2].

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
```

2. https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON

```
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

Ce fichier JSON décrit 6 propriétés, et la propriété *members* est un tableau contenant lui-même des dictionnaires décrivant les héros.

2.2 Manipulation avec Python

Python dispose d'une librairie native `json` qui offre des fonctions pour générer des fichiers JSON et convertir des fichiers JSON en objets.

- `dumps` convertit un objet en une chaîne de caractères formatée en JSON ;
- `dump` convertit un objet en un fichier JSON ;
- `loads` convertit une chaîne de caractères (ou un tableau d'octets) contenant un document JSON en un objet Python ;
- `load` convertit un fichier JSON en un objet Python.

Exercice 6 — Lecture et écriture de fichier JSON

Récupérer le fichier JSON contenant la liste des départements [3], le convertir en dictionnaire Python (fonction `load` car c'est un fichier) et en itérant sur les éléments, générer un dictionnaire des régions contenant leurs départements. Ce dictionnaire sera ensuite converti en fichier JSON (en utilisant `dump`).

Exercice 7 — Sérialisation des classes

Les classes sont des objets complexes que ne connait pas JSON. Pour les sérialiser, il faut donc enregistrer dans un élément compatible avec JSON, comme le dictionnaire, l'ensemble des attributs d'une instance.

3. https://www.data.gouv.fr/fr/datasets/departements-et-leurs-regions/

A partir de l'exemple de fichier JSON donné plus haut (récupérable sur le site), créer une classe représentant un super-héros. Implémenter des méthodes permettant de récupérer sa représentation sous forme de dictionnaire et en JSON.