Compte rendu SAE 32 Aurélien BESOMBES RT2-FI

Server

I: Importation de modules

```
import socket
import pickle
from threading import Thread
```

- **socket** → module principal du SAE
- **pickle** → pour sérialiser les données pour un envoie plus simple
- **thread** → pour gérer la connexion de plusieurs client au serveur

II: Création d'un jeu de donné à partir d'une class

```
class Movie():
    def __init__(self,id,nom,date):
        self.id = id
        self.nom = nom

    def export_dico(self):
        output = {
            'IDmovie': self.id,
            'nom': self.nom,
        }
        return output

f1 = Movie('1','oui','20 janvier 2003')
f2 = Movie('2','non','20 janvier 2003')
f3 = Movie('3','dd','20 janvier 2003')
```

III: Création de variable

```
passwd = [["admin","mdp"],["1","1"],["2","2"]]
cores = {'1':f1,'2':f2,'3':f3}

localIP     = ""
localPort   = 5890
bufferSize  = 1024

TCPServerSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
TCPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
TCPServerSocket.bind((localIP, localPort))
list_of_clients = []

print("TCP server up and listening")
print("----------------------------")
TCPServerSocket.listen(100)
check = True
```

- **passwd** → liste comprenant les utilisateurs & mot de passe pour pouvoir utiliser les commandes sur le serveur, on devrait stocker ce genre d'information dans une base de données, mais pour une question de simplicité, on utilise une simple variable.
- **cores** → lien entre un nom (str) et un object dans le jeu de donné
- **TCPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)** → utilisé par le module `socket` pour libérer le port utilisé après fermeture du programme
- **list_of_clients** → permet d'enregistrer les clients actuellement connectés au serveur et stocker des informations sur eux

IV: Après connexion de l'utilisateur

```
def clientthread(user):
    conn = user['conn']
    addr = user['addr']
    passok = user['p']
    temp = 3
    global check
    while check == True:
        if passok != False:
```

```
            try:
                data = conn.recv(1024).decode()
                if str(data).split(':')[0] == "SEND":
                    com = str(data).split(':')[1]
                    data_send = pickle.dumps(cores[com])
                    conn.send(data_send)
                elif str(data) == "NB":
                    nb = f"Nombre de connecté au server: {len(list_of_clients)}"
                    conn.send(str(nb).encode())
                elif str(data) == "END":
                     print('Connection down:', addr)
                     conn.close()
                     list_of_clients.remove(user)
                else:
                     print("Command not found")
                     try:
                          conn.send("NO".encode())
                     except:
                          conn.close()
                          list_of_clients.remove(user)
            except:
                pass
        else:
                try:
                    data = conn.recv(1024).decode()
                    if str(data).split(':')[0] == "UP":
                        userps = str(data).split(':')[1]
                        password = str(data).split(':')[2]
                        for ele in passwd:
                            if ele[0] == userps and ele[1] == password:
                                passok = True
                        if passok == True:
                            conn.send("PASSOK".encode())
                        else:
                            temp -= 1
                            if temp == 0:
                                conn.send("EC".encode())
                                conn.close()
                                list_of_clients.remove(user)
                                break
                            else:
                                conn.send("ECHEC".encode())
                    else:
                        temp -= 1
                        if temp == 0:
                            conn.send("EC".encode())
                            list_of_clients.remove(user)
                            conn.close()
                            break
                        else:
                            conn.send("ECHEC".encode())

                except:
                    conn.close()
                    list_of_clients.remove(user)
```

- **temp** correspond au nombre de tentative de connexion autorisé avant d'être déconnecté automatiquement.
- Cette partie est séparée en deux cathégorie, en haut pour quand on est identifié et celle d'en bas ou on ne l'ai pas (if & else). Dans les deux cas, on effectué un try pour éviter de faire stopper le serveur en cas d'erreur imprévus. Dans la partie non identifiée, l'utilisateur qui essaye de ce login se voit déconnecter.
- Pour s'authentifier, il faut utiliser la commande `UP:<user>:<password>`. Le problème étant que le programme n'accepte pas le caractère `:` dans le user ou le mot de passe.
- La commande principale et l'envoie d'un objet qui se fait par `SEND:<nom>`, de la même manière, le nom de l'objet ne doit pas avoir de `:` dans son nom. Le `<nom>` est référencer dans la variable `cores`.
- Pour réglé le problème des `:`, il faudrait le remplacer par un simple espace. (Non fait ici, car l'erreur a été vu lors de l'écriture du rapport)

V: Lancement du code pour chaque connexion d'utilisateur

```
while check == True:1
    conn, addr = TCPServerSocket.accept()
    user_add = {'conn':conn,'addr':addr,'p':False}
    list_of_clients.append(user_add)

    print (addr[0] + " connected")
```

```
    start = Thread(target=clientthread,args=[user_add])
    start.start()

TCPServerSocket.close()
```

- Cette dernière bouble `while` est utilisé pour détecter une connection aux serveur et lancé un thread pour cet utilisateur avec la fonction vu au dessus.
- De plus, l'utilisateur est stocké dans une liste que l'on réutilisera dans le code pour connaître le nombre de connexion simultané & avoir les informations de connexion.

Client

```
import socket
import pickle

class Movie():
    def __init__(self,id,nom,date):
        self.id = id
        self.nom = nom

    def export_dico(self):
        output = {
            'IDmovie': self.id,
            'nom': self.nom,
        }
        return output

TCP_IP = "192.168.64.53"
TCP_PORT = 5890
bufferSize = 1024

TCPClientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
TCPClientSocket.connect((TCP_IP, TCP_PORT))
check = True
while check == True:
    message = input(" -> ")
    TCPClientSocket.send(str.encode(message))

    if message == "END":
        TCPClientSocket.close()
        break

    data = TCPClientSocket.recv(2048)

    try:
        if data.decode() == "NO":
            print("Command not found")

        elif data.decode() == "EC":
            print('Time Out par server')
            TCPClientSocket.close()
            break

        elif data.decode() == "ECHEC":
            print('USER/PASSWS FALSE')

        elif data.decode() == "PASSOK":
            print('Pass OK')

        elif data.decode() != "NO":
            print(data.decode())


    except:
        print('Received from server')
        received_grades = pickle.loads(data)
        print(received_grades.export_dico())

TCPClientSocket.close()
```

- Dans la partie client, on retrouve la création de variable, la class.
- Cette fois si, il y a seulement une boucle while qui nous permet d'envoyé un message.

- Le client fonctionne par rapport au message que le serveur lui renvoie. Exemple, si le serveur envoie `NO`, le client comprendra que la commande n'existe pas côté serveur.
- Étant donné que les objet sont envoyé directement en binaire et ne peuvent pas être décodé par python, mais seulement par pickle, on effectue un test avec try et si l'erreur de décodage arrive, l'except prend le relais et comprend qu'il s'agit de donné binnaire destinée à Pickle.