TP n°3 *TCP, client/serveur*

Exercice 1. `count-server.py`

Vous allez refaire le service que je vous ai montré en classe : le service de "compteur". Le serveur contient la base de données la plus simple qui soit : un compteur. Les clients peuvent intéragir avec le serveur, et au travers de requètes, consulter la valeur courante du compteur, ou la modifier.

Dans un premier temps, nous nous limiterons au protocole suivant :

- le client envoit une requète `get` et le serveur lui renvoit une réponse `val` N ou N est la valeur courante du compteur.
- le client envoit une requète `quit` et le serveur lui renvoit une réponse `quit` et ferme la connection.

count-server.py

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

def server(port):
    sock = socket.socket()
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(("0.0.0.0", port))
    sock.listen(10)
    while True:
        cli, addr = sock.accept()
        session(cli)

COUNTER = 0

def session(cli):
    global COUNTER
    f = cli.makefile(mode="rw")
    while True:
        cmd = f.readline().strip()
        if cmd == "get":
            f.write("val␣%d\n" % COUNTER)
            f.flush()
```

```
        elif cmd == "quit":
            f.write("quit\n")
            f.flush()
            break
        else:
            f.write("err\n")
            f.flush()
    f.close()
    cli.shutdown(socket.SHUT_RDWR)
    cli.close()
```

Vous pouvez déjà tester ce server avec `netcat` :

```
$ netcat localhost 4444
get
val 0
quit
quit
```

Exercice 2. count-client.py

Voici un client qui refait le test précédent :

count-client.py

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

def client(host, port):
    sock = socket.socket()
    sock.connect((host, port))
    f = sock.makefile(mode="rw")
    f.write("get\n")
    f.flush()
    print(f.readline(), end="")
    f.write("quit\n")
    f.flush()
    print(f.readline(), end="")
    f.close()
    sock.shutdown(socket.SHUT_RDWR)
    sock.close()

client("localhost", 4444)
```

Exercice 3. Version orientée-objet

Copiez `count-server.py` en `count-server2.py`, puis refactorisez le code pour faire apparaitre une classe `Server` et une classe `Session`. La compteur qui était précédemment global, doit maintenant être un attribut du `Server`.

Testez cette nouvelle version avec `netcat`.

Exercice 4. Version orientée-objet multi-threadée

Copiez `count-server2.py` en `count-server3.py`, puis faites en sorte que chaque requète soit traitée sur son propre thread. Vous allez avoir besoin de :

```
from threading import Thread
```

Exercice 5. Version orientée-objet multi-threadée + 1 méthode par requète

Copiez `count-server3.py` en `count-server4.py`, puis faites en sorte que chaque requète soit traitée par une méthode spécifique de la session. Vous allez avoir besoin d'adapter votre code, Voici un rappel du schéma qui a été vu en cours :

```
line = self.file.readline().strip()
if not line:
    break
cmd, *args = line.split()
method = "command_%s" % cmd.lower()
getattr(self, method)(args)
```

Exercice 6. Extensions du protocole

Étendez le protocole avec les opérations suivantes :

`incr` incrémente le compteur et retourne la réponse `ok`.

`decr` décrémente le compteur et retourne la réponse `ok`.

`add` *N* ajoute *N* au compteur et retourne la réponse `ok`.