Projet Chat

V. Python

Diana Heddadji

PARIS VIII - UFR MITSIC / L2 Informatique

15 juin 2018

Sommaire

Le but de ce projet est de parvenir à réaliser un chat en groupe de cinq.

Chaque membre du groupe choisira un langage de programmation différent afin de créer un client et un serveur en suivant un protocole précis.

protocole.md : Protocole

Dans la partie python du projet, il y a deux fichiers :

- **client.py**
- **serveur.py**

On commence par importer notre socket avec :

import socket

Pour créer notre socket, il est nécessaire de faire un appel au constructeur **socket**.

On aura besoin des deux paramètres suivants s'il s'agit d'une connexion **TCP** :

- **socket.AF-INET** : la familles des adresses internet ;
- **socket.SOCK-STREAM** : Pour le protocole **TCP**, **SOCK-STREAM** est le type du **socket**.

main-connection = socket.socket(socket.AF-INET, socket.SOCK-STREAM)

Pour connecter le **socket**, on utilisera la méthode **bind** dans le cas où le serveur attend des clients. Cette dernière prend le tuple **(nom-hote, port)**.

main-connection.bind((", 2222))

Pour faire écouter notre **socket**, il faut lui renseigner le nombre maximum de connexions qu'il peut recevoir sur le port sans les accepter. Ceci ce fait par la méthode **listen** qui prend ce nombre en paramètre.

Pour accepter une connexion venant du client, l'usage de la méthode **accept** est nécessaire afin de bloquer le programme tant que personne ne s'est connecté.

Cette méthode retourne deux informations :

- La première est le **socket** connecté qui vient de se créer qui nous permettra par la suite de dialoguer avec notre client.
- La seconde est un tuple comportant l'adresse **IP** ainsi que le port de connexion du client.

connection-client, connection-info =
main-connection.accept()

Pour créer ensuite le client, le principe est le même que celui du **socket** :

```
import socket
connection-server = socket.socket(socket.AF-INET,
socket.SOCK-STREAM)
```

On connecte ensuite le client avec la méthode **connect**. Celle-ci prend un tuple en paramètre comportant le nom d'hote et le numéro du port identifiant le serveur auquel on veut se connecter.

```
connection-server.connect(('localhost', 2222))
```

Le serveur et le client étant désormais connectés, cela signifie que la méthode **accept** ne bloque plus le programme étant donné qu'elle vient d'accepter la connexion du client.

On peut afficher l'adresse **IP** et le **port** du client avec **print(connection-info)** du côté du serveur.

L'adresse IP vaut **127.0.0.1** il s'agit de l'adresse **IP local** de la machine est donc **localhost** redirige vers cette adresse **IP**.

Pour faire communiquer nos sockets, il suffit d'utiliser les méthodes **send** et **recv**.

Au niveau du serveur cela se fera comme suit :

client.send(b"ok")

- **send** : retourne le nombre de caractères envoyés
- **recv** : prend ce que retourne **send** c'est-à-dire, le nombre de caractères à lire.

Côté client, on réceptionnera le message que l'on vient d'envoyer.

Étant donné qu'on ne sait pas à l'avance le nombre de caractères qu'on recevra, on lui donne conventionnellement la valeur 2000.
Si le message comporte plus de caractères alors on récupère le reste après.

Côté client, ceci se fera comme suit :
msg-received = connection-server.recv(2000)

À chaque fois que le serveur reçoit un message, il envoie un accusé de réception : **"OK"**.

Côté client, on peut remarquer l'utilisation des méthodes de **str** : **encode** et **decode** ;

encode sert à partir d'un nom d'encodage **(Utf-8)** de passer un **str** en chaîne **bytes** et **decode** permet de faire exactement l'inverse.

Les informations transmises par **send** et **recv** sont des chaînes de **bytes**, pas des **str** et c'est **encode** et **decode** qui permettent de traduire les messages reçus et envoyés.

Du côté de notre serveur, on peut voir la présence du module **select**, celui-ci va interroger plusieurs clients dans l'attente d'un message à réceptionner, sans mettre en pause le programme.

Select va donc écouter sur une liste de clients et retourner au bout d'un certain temps.

Ce qui est retourné est la liste des clients qui ont un message à réceptionner.

On utilisera la méthode **select** qui prend trois ou quatre arguments et en retourne trois.

Dans l'ordre, les paramètres de **select** sont :

- **rlist** : la liste des sockets en attente d'être lus ;
- **wlist** : la liste des sockets en attente d'être écrits ;
- **xlist** : la liste des sockets en attente d'une erreur ;
- **timeout** : le délai pendant lequel la fonction attend avant de retourner.

On créer une liste **connected-client** afin d'y mettre des **sockets** de façon à ce que **select** les surveille et puisse retourner dès qu'un **socket** est prêt à être lu. Ainsi, le programme ne bloquera pas et pourra recevoir des messages de plusieurs clients.

En précisant notre **timeout**, **select** retournera au bout du temps en secondes que l'on aura indiqué **(ici 0.05)**, ou si un **socket** est prêt à être lu.

select renvoie trois listes **rlist**, **wlist** et **xlist**.

read-client, list1, list2 = select.select(connected-client, [], [], 0.05)

Cette instruction va écouter les sockets contenus dans la liste **connected-client**.

Enfin pour fermer la connexion, il suffit d'appeler la méthode **close** sur le **socket**.

Côté client :

connection-server.close()

Côté serveur :

main-connection.close()

Le serveur tourne jusqu'à recevoir le message **"BYE"**.