Compte rendu de projet

Pierre VAUDRY

Jessy JOSE

IPSA - 2020

Introduction

Réseau informatique

Un réseau informatique (en anglais, data communication network ou DCN) est un ensemble d'équipements reliés entre eux pour échanger des informations. Par analogie avec un filet (un réseau est un « petit rets », c'est-à-dire un petit filet1), on appelle nœud l'extrémité d'une connexion, qui peut être une intersection de plusieurs connexions ou équipements (un ordinateur, un routeur, un concentrateur, un commutateur).

Indépendamment de la technologie sous-jacente, on porte généralement une vue matricielle sur ce qu'est un réseau.

De façon horizontale, un réseau est une strate de trois couches : les infrastructures, les fonctions de contrôle et de commande, les services rendus à l'utilisateur. De façon verticale, on utilise souvent un découpage géographique : réseau local, réseau d'accès et réseau d'interconnexion.

Objectif

L'objectif de ce projet est de découvrir le réseau à travers le langage de programmation python. Le réseau étant une notion très importante, le programme s'orientera sur la base la plus commune du fonctionnement en réseau. Le fonctionnement de base étant basé sur : client – serveur.

Projet

Afin de découvrir la notion de réseau sur le principe de client-serveur, notre projet se base sur l'idée d'une messagerie instantané sécurisé basé sur le modèle d'un serveur-client donné en amont.

Pourquoi python ?

Python est un langage haut niveau, facile à prendre en main et qui est applicable dans de nombreux domaine d'application (autant dans des projet scientifique ou informatique que des projet basic et pratique). Ce langage flexible s'intègre parfaitement à la notion de réseau grâce au module socket.

Le module socket offre une large utilisation en réseau et permet de faciliter l'écriture d'un programme client – serveur. Pour plus d'information sur le module en lui-même ou sur python, nous vous invitons à lire la documentation fournie par python et sa communauté.

Présentation programme

Programme serveur

Le programme serveur est un programme contenant l'ensemble des instruction, fonctions et processus côté serveur. Le programme serveur est constitué de plusieurs modules, fonction et processus.

Nous avons choisi de couper le serveur en plusieurs fonctions et processus afin de l'optimiser et permettre l'utilisation de code dans d'autre programme.

Le programme permet la réception des requêtes de plusieurs clients de façon quasiment simultanée. Et leur attribut une réponse. On utilise des bases de données non-sql afin de gérer les données côtés serveur (une utilisation de SqlLite est possible mais nous devions utiliser le moins de module). Les données sont contenues dans des fichiers .txt (un fichier pour les messages client et un fichier pour enregistrer les connexions avec les différents clients).

Le serveur va compléter la base de données du serveur à partir des requêtes reçues par les nombreux clients. La réponse sera alors créée à partir de la base de données obtenue et faite tout au long de l'exécution du code.

Programme client

Le programme client est similaire au programme serveur. Mais il n'y a pas de création d'instance (uniquement côté serveur). Il y'a uniquement une partie connexion (s.connect()).

Le programme permet l'envoie d'un message rédigé par l'utilisateur. Puis réceptionne la réponse donnée par le serveur. Il y'a un algorithme de chiffrage fait de bout en bout des clients.

Seuls les clients ont les clefs de chiffrement et déchiffrement. La partie message est l'unique partie chiffré de la requête, le reste étant des données informatives sur le client lui-même (information tel que la date de rédaction du message, l'id du client qui envoie, les informations propres au socket client tel que l'ip et le port qui lui est propre).

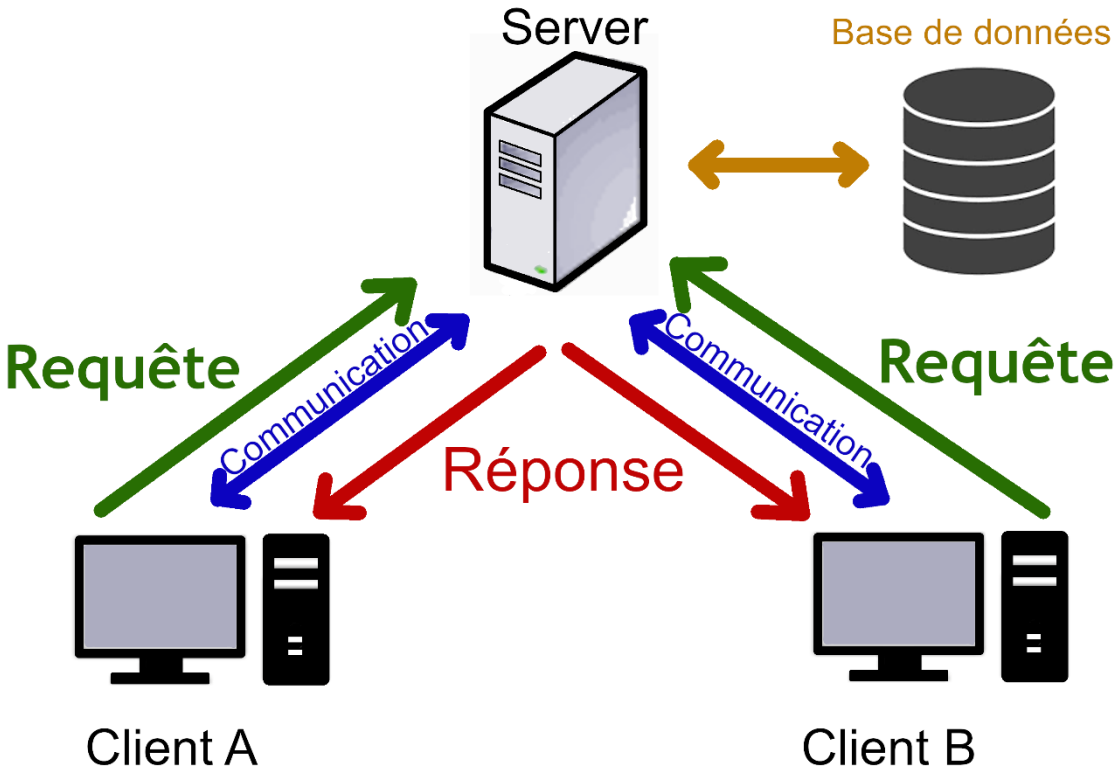
Une fois le serveur et le client connecté, il n'y a que le client qui peut éteindre le serveur (le serveur relai juste la fermeture des connexions avec les autre clients)

Exemple de dialogue entre les clients et le serveur.

```
Client A => Chiffre : Bonjour Jacque, c'est George
(après chiffrement)=> Envoie : 2020-11-20T12:28:53.316399|[CLIENT
127.0.0.1:52420]@vox)oPZEOmf'P[ZEfI[j-E0[oZb[

Serveur => Reçoit : 2020-11-20T12:28:53.316399|[CLIENT
127.0.0.1:52420]@vox)oPZEOmf'P[ZEfI[j-E0[oZb[
Serveur => Traite ce qu'il a reçu
Serveur => Envoie : {du vide car il n'y avait pas de message enregistré avant}
```

```
Client A => Reçoit : {du vide car il n'y avait pas de message enregistré avant}
Client A => Déchiffre : {du vide car il n'y avait pas de message enregistré avant}

_____________________________________

Client B => Chiffre : Bonjour George, comment vas-tu ?
(Après chiffrement)=> Envoie : 2020-11-20T12:35:07.286946|[CLIENT
127.0.0.1:52421]@vox)oPZE0[oZb[ZEfoyy[x-ECmj4-PE_

Serveur => Reçoit : 2020-11-20T12:35:07.286946|[CLIENT
127.0.0.1:52421]@vox)oPZE0[oZb[ZEfoyy[x-ECmj4-PE_
Serveur => Traite ce qu'il a reçu
Serveur => Envoie :[CLIENT 127.0.0.1:52420]@vox)oPZEOmf'P[ZEfI[j-E0[oZb[

Client B => Reçoit :[CLIENT 127.0.0.1:52420]@vox)oPZEOmf'P[ZEfI[j-E0[oZb[
Client B => Déchiffre : Bonjour Jacque, c'est George
```

Schéma des communications entre le client et le serveur



Notre projet comporte deux types d'interaction formant la communication ou canal.
    Interaction 1 : la requête envoyée par le client au serveur. A ce moment-là le client envoie le message déjà chiffré
    au serveur et le serveur reçoit
    Interaction 2 : La requête 'Réponse' est l'envoie des données contenues et stockés par le serveur vers le client

Il y'a bien 2 Interaction pour une communication entre client/serveur.
Donc afin de communiquer, le schéma de communication se répète. Un client envoie puis reçoit et attend de pouvoir
recommuniquer avec le serveur une fois qu'il sera disposé à communiquer (une fois que l'autre client aura fini sa
communication avec le serveur).

Exemple des Interaction de façon générale :

    Démarrage Serveur
    Démarrage Client A puis connexion
    George utilise le client A
    Démarrage Client B puis connexion
    Jacque utile le client B

    Communication Client A / Serveur

    Client A => envoie un message donnée par l'utilisateur (déjà chiffré en amont)
    Serveur => Reçoit les donnée du Client A puis les traites (il les stocks et les met dans un format utilisable)

    Serveur => Envoie le format utilisable au Client A
    Client A traite l'information

    _____

    Communication Client A / Serveur

    Client B => envoie un message donnée par l'utilisateur (déjà chiffré en amont)
    Serveur => Reçoit les donnée du Client B puis les traites (il les stocks et les met dans un format utilisable)

    Serveur => Envoie le format utilisable au Client B
    Client B traite l'information

    _____

    Le dernier message présent sur le serveur étant celui du Client A, le Client B a donc bien le message de son ami

Donnée du serveur

Server_data

Server_data est la donnée permettant de compter le nombre de client qui se connecte au serveur et leur permet ainsi d'avoir un ordre de passage définit par rapport à leur position.

Server_data est un dictionnaire comportant une clé (key => 'client') et une valeur (Integer) qui lui est attribuée (value => 0).

Base de données du serveur

Le serveur procède une base de données non-sql qui lui est propre. En effet, cette base de données enregistrer toute les requêtes clients (quel qu'ils soient) afin de les utiliser dans un contexte et un ordre bien précis. Cette base de données permet de rendre possible l'échange de données entre les clients sans qu'ils n'interagissent directement avec la base de données.

Donnée du client

client_data est la donnée permettant de renseigner les information relative au client.

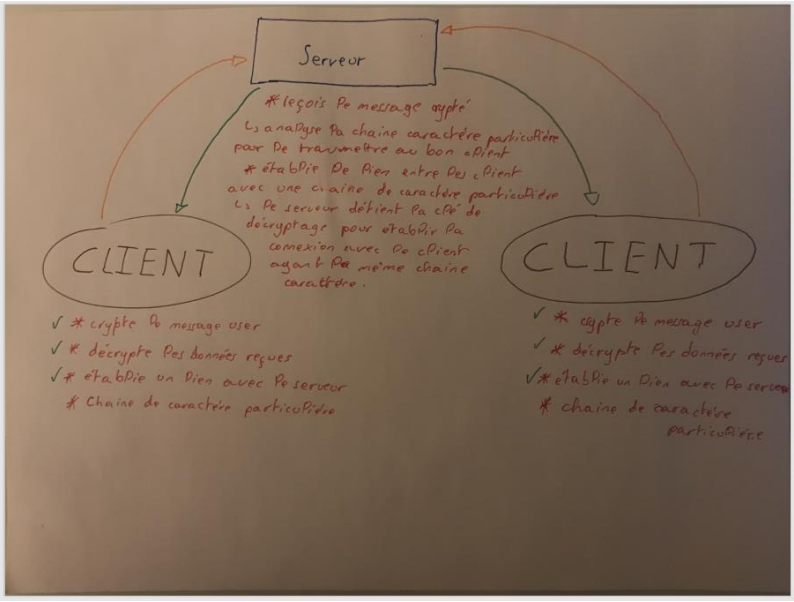
client_data est un dictionnaire comportant plusieurs clés (key) et des valeurs (Integer) qui lui sont attribuées (value).

Programme

Notre idée était de faire une boite de dialogue instantané entre deux ordinateurs (clients) comme les logiciels (outils) qu'on trouve actuellement (à savoir Skype, Discord, Teams, WhatsApp, Messenger…).

Avant la partie programmation, il a fallu mettre en place un Brain Storming du projet. Ce Brain Storming nous a permis de trouver et articuler le fonctionnement du projet.

Ces préparatifs ont abouti à un schéma initial qui affichait les prémices du projet.



Une fois l'idée choisie, nous avons mis en place le projet. Pour une meilleure organisation, nous avons mis le projet sur Github afin de garder une trace des Update et amélioration (ou pour revenir à une version antérieur fonctionnelle). La plateforme nous a permit de coder en toute circonstance et de façon organisé. La documentation, le ReadMe et le reste du projet a été fait progressivement au cours du temps. La partie la plus compliqué étant la partie réseau (voir difficulté, correctif possible ou prévue).

Conception et architecture [programme serveur]

Conception du programme côté serveur

Une fois le projet lancé nous avons pris la base du programme fournie en tant qu'exercice de découverte des notions de réseau avec python.

Cependant, nous nous sommes approprié le code et nous l'avons modifié afin de correspondre au maximum au contrainte de notre projet.

Le programme présente 3 fonctions particulières :

- Les procédures principales (fonction ne retournant rien) : elles constituent le cœur du programme en permettant la connexion réseau ou en implémentant le serveur à proprement dit
- Les procédures ou fonctions secondaires (ou d'application) : elles permettent de répondre à un besoin spécifique (exemple : chiffrer le texte utilisateur), elles sont là pour effectuer une tache précise et permet de soulager le code principal du serveur.
- Les autres procédures ou fonctions : cette partie regroupe les procédures d'affichage, ou non essentiel au programme (ou alors en cours de création donc non utilisé)

Afin de se repérer, les fonctions dites principales sont en bas du programme tandis que les fonctions moins essentielles sont vers le haut (choix d'ordre personnel).

Architecture du programme côté serveur

Le programme est de la forme suivante :

```
> Security-Message-Communication
|- Server
        |- log.txt
        |- log_connection.txt
        |_ server.py
```

Le programme du serveur est server.py.

Une fois lancé il va interagir avec deux fichiers utilisés comme base de données (non-sql). Le premier fichier avec lequel il va interagir c'est log_connection.txt. Le second sera utilisé uniquement s'il y a dialogue ou sollicitation du client au serveur.

Le programme est composé de fonctions et de procédures afin de le rendre modulable, de tester chaque nouvelle fonctionnalité ou code, et de pouvoir réutiliser des procédures utiliser dans autre endroit du programme.

Chaque fonctions/procédures possèdent une documentation qui lui ai propre.

Le programme commence à la fin via une condition de test.

Conception du programme côté client

Tout comme le programme côté serveur, les codes des clients ont été fait à partir de l'exercice donnée sans le reprendre totalement.

La conception et l'architecture du client est identique à celle du programme serveur. L'ordre des fonctions est identique.

La partie client est plus difficiles au niveau de sa conception car les autorisations liées à l'environnement Windows peuvent compliquer les essais entre deux pc sur des réseaux distant.

Architecture du programme côté serveur

Le programme est de la forme suivante :

> Security-Message-Communication
|- CleintX
        |_ client.py

Le programme du serveur est client.py.

On aurait pu sortir certaine fonction en tant que module, mais comme le projet n'est pas volumineux et qu'un module peut être plus difficile à mettre en place, nous avons procédé de la même façon que pour le serveur.

Programme Serveur

Le programme commence ligne 401 :

```
if __name__ == '__main__':

    # Give basic and native documentation in console
    documentation()

    # Run the program
    run()
```

Ce bout de code permet, non seulement d'amorcer le serveur (on appel la fonction qui va démarrer la fonction principale du serveur), mais aussi de lancer une documentation à l'écran côté serveur.

La boucle de la procédure run() permet de maintenir la connexion à l'état True (sinon elle s'arrête d'elle-même selon comment elle est programmée).

```
while True:
    connection_server()
```

La procédure

```
connection_server()
```

est le cœur du serveur. En effet, la procédure comporte le code pour ouvrir une communication en réseau, ainsi que les autres fonctionnalités qui lui sont rattachées.

Cette procédure est différente de celle donnée par l'exercice. En effet, nous avons choisi de faire une boucle while avec un système de passage d'erreur en cas de soucis.

D'autres boucles permettent (boucle for) de faire un système de trie de client.

Parmi les boucles permettant l'écoute lors de la connexion et l'attente des clients on a :

```
for connection in wait_connections:
    connection_client, info_connection = connection.accept()
    client_connected.append(connection_client)
    print("Prosition : ", turn , " | Client : ", connection_client)
    turn = turn + 1
    log_connection(connection_client)
```

pour l'écoute de la réception et l'envoie :

```
for client in read_client:
    msg_recv = client.recv(1024)

    msg_recv = msg_recv.decode()

    process_server(msg_recv)
```

Si l'exécution de la boucle while se termine correctement on a une boucle qui permet d'arrêter les clients de la liste d'attente :

```
for client in client_connected:
    client.close()
```

Parmi les fonctions dites secondaires, nous avons :

```
process_server(data)
```

Fonction qui permet de traiter la réponse donnée par le client actuellement connecté.

```
list_log()
```

Fonction qui permet de couper le message reçu sous forme de liste utilisable par le serveur (à savoir une liste qui comporte : date de transfère message, identifiant et message chiffré reçu.

Toutes les autres fonctions, procédures, permettent la conversion, l'utilisation des données ou de l'affichage côté serveur.

Programme Serveur

Le programme commence ligne 401 :

```
if __name__ == '__main__':

    # Give basic and native documentation in console
    documentation()

    # Run the program
    run()
```

Ce bout de code permet, non seulement d'amorcer le client (on appel la fonction qui va démarrer la fonction principale du client), mais aussi de lancer une documentation à l'écran côté client.

La procédure run() permet de démarrer le client.

La procédure

```
connection_client(HOST, PORT)
```

est le cœur du client. En effet, la procédure comporte le code pour ouvrir une communication en réseau, ainsi que les autres fonctionnalités qui lui sont rattachées.

Cette procédure est différente de celle donnée par l'exercice. En effet, nous avons choisi de faire une boucle while avec un système de passage d'erreur en cas de soucis.

```
send_msg = b""
while send_msg != b"/stop":
    # msg contains message given by user
    msg = input("> ")
```

Cette boucle permet de prendre le message de l'utilisateur mais de stopper net la connexion en cas de commande via la saisie utilisateur.

Comme fonction secondaire nous retrouvons une fonction similaire au serveur à savoir :

```
process_response(data, data2, messager)
```

Fonction qui permet de traiter la réponse donnée par le serveur avec lequel il est actuellement connecté.

Toutes les autres fonctions, procédures, permettent la conversion, l'utilisation des données ou de l'affichage côté client. (Exemple : la fonction de chiffrage est utilisée dans la procédure principale mais non essentielles)

Pour plus de renseignement, merci de lire la documentation fournie avec le projet (ReadMe.md et documentation dans le code).

Utilisation du projet

Courte histoire du projet

> *Imaginons que Mr E. Snowden soit la cible d'attaque d'hacker russe. Il doit communiquer impérativement avec des personnes à l'autre bout du monde de façon sécurisé et anonyme. C'est là que notre projet est utile !!!*
> En effet, le projet permet de discuter de façon sécurisée avec une autre personne quelque soit la distance les séparant.

Afin de pouvoir jouir pleinement du projet, il faut un environnement python configuré avec les

Modules : datetime, select, socket.

Une fois ces modules correctement installés dans votre environnement python,

Lancé votre IDLE ou autre programme similaire permettant d'ouvrir et lancé python.

Python 3.5 ou supérieur est nécessaire pour le bon fonctionnement du projet

> sur console cmd ou PowerShell (ou IDLE):

--> console 1 (console serveur):

PS C:\WINDOWS\system32> cd /le/chemin/de/votre/server

PS C:\le\chemin\de\votre\server> python server.py

Le serveur est prêt à recevoir les messages qui lui seront envoyé afin de les stocker et

Transmettre à l'autre client

--> console 2 et 3 (console client):

PS C:\WINDOWS\system32> cd /le/chemin/de/votre/clientA|clientB

PS C:\le\chemin\de\votre\clientA|clientB> python clientA|B.py (nommé client.py)

Le client A se connecte au serveur et est en attente

Une fois que le client B s'est connecté, le dialogue peut commencer.

> sur Pycharm :

Exécuté les fichier python dans l'ordre suivant :

> serveur.py --> client A --> client B

Le serveur sera sur écoute, le client A puis le client B seront connecté au serveur et prêt pour emploi.