Compte rendu du « mini » projet réseau.

NASH DEBORAH

DESMESURE ARTHUR

TMA1_7

Introduction :

Le but de ce projet était avant tout de recréer de manière simplifiée un serveur de type IRC. Pour ce faire nous avions à notre disposition le langage python3.

Le but d'un serveur IRC est de :

1) Pouvoir accueillir plusieurs clients

2) Que ces clients puissent créer des « canaux » de discussions

3) Que ces canaux soient accessibles à tous les clients, sauf si décidé autrement

4)Que les clients, à l'arrivée sur le serveur puissent choisir leurs pseudos.

5) Que les clients puissent rentrer et sortir de canaux.

Commencement :

Après analyse du sujet, le problème principal qui nous est apparu à été que nous ne pouvions, et ne savions pas comment faire pour utiliser plusieurs ordinateurs pour concevoir et tester notre programme. Nous avons donc dû nous adapter, et au lieu d'utiliser les adresses IP comme moyen d'authentification des différents clients, nous avons utilisé les « sockets ».

Selon OpenClassroom un socket est « *un point de terminaison d'une*

*communication bidirectionnelle, c'est-à-dire entre un client et un serveur en cours d'exécution sur un **réseau** donné. Les deux sont liés par un même numéro de port TCP de sorte que la couche puisse identifier la demande de partage de données »*

En d'autres termes, c'est le « câble » virtuel qui s'ouvre lors d'une connexion entre un client et un serveur, qui permet aussi d'identifier ladite connexion.

Dans le but de pouvoir utiliser les sockets il à fallu que nous mettions en place des dictionnaires, voir même des dictionnaires de listes. En effet, lorsqu'un client se connecte au serveur, on récupère son socket, et on le sauvegarde dans un dictionnaire. Ensuite le client est prié de rentrer un pseudo. Nous associons ce pseudo au socket de sorte à pouvoir revenir le chercher quand nous en avons besoin.

V.0 :

Lors de la V0 de ce projet, il nous était demandé de mettre en place les caractéristiques de bases de ce projet. Pour réussir nous avons fait usage des dictionnaires

-Le client peut rejoindre le serveur

→ En lançant le programme, le client est automatiquement inscrit dans le dictionnaire « DictClient »

-Avec la commande « /bye » il peut le quitter

→ On supprime le nom du client du dictionnaire et on rompt la connexion du socket

→ Pour faire ça il nous à suffit de faire un « socket.close » qui permet tout simplement de fermer le socket dans lequel se trouve le lanceur de commande.

-La commande « /join » permet de rejoindre un canal de discussion, et si ce dernier n'existe pas déjà, de le créer avant de le rejoindre.

→ Pour réaliser cette commande nous avons dû nous servir de dictionnaire (en l'occurrence « Dictchannel ». La fonction JOIN demande en paramètre une chaine de caractères. Lorsque /JOIN est invoqué notre programme rajoute dans ce dictionnaire le nom du canal que l'utilisateur vient de rentrer. Et en même temps lie le pseudo de l'utilisateur qui a appelé /JOIN au canal (dans le Dictchannel).

-Et avec « /leave » il peut le quitter

→ Le principe est l'inverse du JOIN, sauf que cette fois le client n'a pas besoin de préciser le nom du canal. En effet on récupère simplement ce dernier car il est lié au NICK. Il ne nous reste plus maintenant qu'a détruire le canal lorsque la dernière personne présente /LEAVE.

-Une commande « /List » qui affiche tous les canaux déjà créer.

→ On se sert encore une fois de notre dictionnaire de canaux : on le parcourt simplement et on affichant son contenu.

-Un « /who » une commande qui permet de lister et d'afficher les autres personnes qui se trouvent dans un canal

→ Toujours en parcourant le dictionnaire de Canaux, on peut afficher simplement les clients présents d'un canal.

- « /kick » permettra de sortir de force un utilisateur d'un canal.

→ Cette dernière à été un peu plus complexe que les autres étant donné qu'il fallait prendre en compte que ce n'était pas la personne lançant la commande qu'il fallait expulser. Ainsi /kick prends en paramètre un utilisateur et, après avoir vérifié que le lanceur était un « admin » (ici le créateur du canal) supprime l'utilisateur ciblé.

Ce qui ne marche pas :

- ➔ Par étourderie nous avons retiré de notre code les messages de types « broadcast » qui nous permettaient d'envoyer un message à tous les clients du serveur, et nous n'avons pas eu le temps de l'y replacer.

- ➔ Nos clients peuvent se connecter à un canal, regarder qui s'y trouve, partir, et même expulser d'autres personnes, peuvent envoyer des messages mais ils ne le reçoivent qu'après avoir entré une nouvelle commande.
- ➔ Nous avons passés beaucoup de temps sans comprendre la fonction select.select, qui était la base des connexions multiples. Et parfois notre méthode de connexion peut nous poser des problèmes.

Ce qui nous a posé problème :

- ➔ Comme dit plus haut select.select, et les connexions multiples, que nous n'avons réussi à maitriser que très tard.
- ➔ L'absence de possibilité de récupérer des arguments passés à une fonction : nous avons dû faire notre fonction pour l'occasion
- ➔ Une erreur inconnue qui ne nous a pas permis d'envoyer des messages à l'intérieur des canaux
- ➔ Nous ne connaissions pas (ou très peu) le Python avant le commencement de ce projet
- ➔ Manipuler des sockets est bien plus complexe que de manipuler des adresses IP
- ➔ Le threading était une partie essentielle du projet afin que les utilisateurs puissent communiquer en « temps réel ». Or c'est très complexe.

Conclusion :

Si nous devions résumer ce projet je pense qu'on dirait qu'il était vraiment intéressant : comprendre certaines subtilités du python, certains enjeux des tchats type IRC ou encore s'immiscer dans l'esprit des concepteurs de ce genre de programmes était vraiment passionnant. Nous ne connaissions pas tout deux le python avant cet exercice, et devoir se plonger dedans la tête la première nous a permis de progresser plus vite que nous n'aurions jamais pu le faire autrement, et en même temps, ce temps passé à apprendre et à nous former à ce langage nous a rapidement désavantagé. Certes le python est un

langage aussi simple que polyvalent, mais il n’en reste pas moins un langage à part entière, avec ses structures de code, ses pièges et ses nuances. Par ailleurs, en regardant sur internet, nous en sommes arrivés à la conclusion que notre projet était un peu particulier. En effet, peu de programmes de Tchat utilisent directement les sockets comme moyen de différenciations des clients. Les programmes amateurs dans ce genre en effet, se basent beaucoup plus sur les adresses IP (Originales et uniques pour chaque client). Pour terminer nous voudrions dire que bien que très incomplet nous sommes fiers de notre travail.

ANNEXE :

Grille d'auto-évaluation

Version 0

Commandes	Score (/10)
connect & disconnect without errors	9/10
login & bye (/BYE)	10/10
list channels (/LIST)	10/10
join & leave channel (/JOIN & /LEAVE)	10/10
user list (/WHO)	10/10
public message	5/10
private message (/MSG)	5/10
rename channel (/REN)	10/10
kick user (/KICK)	8/10

Version 1

Commandes	Score (/10)
change current channel (/CURRENT)	0/10
private message to several users (/MSG)	0/10
change user nickname (/NICK)	0/10

| grant admin privileges (/GRANT) | 0/10 |

| revoke admin privileges (/REVOKE) | 0/10 |

| send & receive file (/SEND & /RECV) | 0/10 |

| history (/HISTORY) | 0/10 |

Bonus (à compléter)

| Commandes | Score (/10) |

| ---------- | ---------- |

| bonus 1 (/CMD1) | |

| bonus 2 (/CMD2) | |

| bonus 3 (/CMD3) | |

| ... | |