Programmation avec des sockets en Python

Exercice 1 Conversation en alternance.

On souhaite écrire une application pour que deux utilisateurs `A` et `B` puissent s'échanger des messages sur Internet. Pour cela, on va écrire deux programmes Python, `server.py` et `client.py` tels que :

- `A` lance le programme `server.py`. Ce dernier implémente un serveur très simple qui attend que `B` se connecte en lançant le programme `client.py`.
- Une fois la connexion établie, l'échange de messages se fera en alternance : `B` commence par envoyer un message, puis `A` lui répond, et on recommence.

On essaiera de gérer proprement les déconnexions ou problèmes réseaux.

Exercice 2 Conversation libre.

On souhaite étendre les deux programmes précédents de manière à ce que la conversation soit libre : on ne sait pas qui va commencer à parler et un utilisateur peut envoyer plusieurs messages avant que l'autre ne réponde. une solution pour cela est d'utiliser un thread supplémentaire (que le thread principal) dans chacun de ces programmes pour gérer la lecture des messages reçus.

Mini-projet : Serveur de discussion

Le but de ce mini-projet est de réaliser un service de discussion en ligne (*Internet Relay Chat* ou IRC) en Python. Il s'agit d'un système client/serveur permettant à des utilisateurs de discuter en direct en s'envoyant des messages. Les utilisateurs peuvent discuter en groupe à travers des *canaux* de discussion, mais également deux-à-deux de manière *privée*.

Le principe est assez simple : des utilisateurs se connectent à un *serveur* IRC en utilisant un programme *client*, tape des commandes et le serveur exécute ces commandes. Le réseau IRC est constitué de serveurs connectés entre eux, sans topologie particulière. Chaque client se connecte à un des serveurs et les commandes (ou messages) qu'il tape sont communiquées par son serveur de rattachement aux autres serveurs, jusqu'aux clients destinataires. Les commandes acceptées par un serveur sont assez nombreuses et tout est décrit dans un protocole décrit dans la RFC1459 : `https://datatracker.ietf.org/doc/html/rfc1459`.

Le guide suivant décrit de manière succinte la grande majorité des commandes disponibles :

`https://fr.wikipedia.org/wiki/Aide:IRC/commandes`

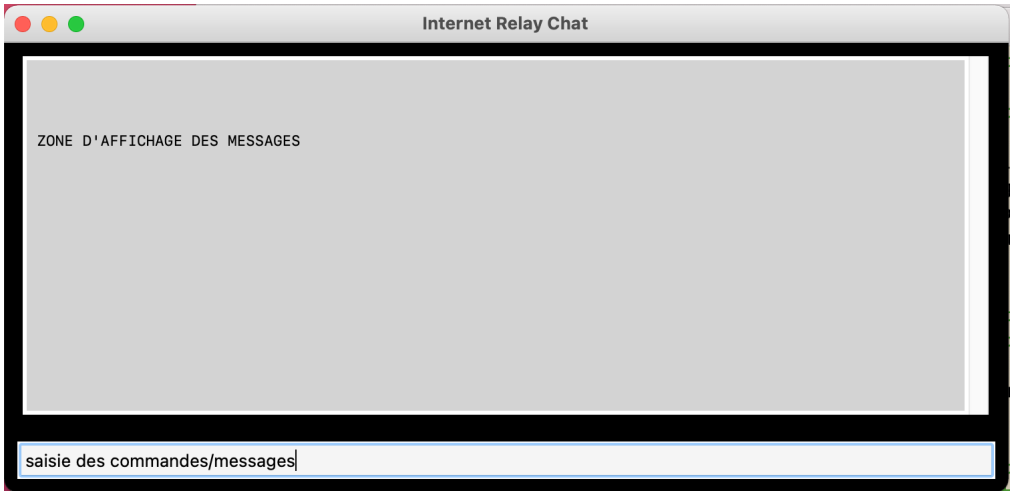
Comme vous pouvez le voir, les commandes ont la forme suivante `/commande <arguments>`, où `<arguments>` représente une liste d'arguments (cette liste pouvant être vide). Dans ce projet, nous allons implémenter un petit sous-ensemble de ces commandes. Chaque client est identifié par un surnom (*nickname*). Les noms de canaux commencent par un symbole `#`.

`/away [message]`	Signale son absence quand on nous envoie un message en privé (en réponse un `message` peut être envoyé). Une nouvelle commande `/away` réactive l'utilisateur.
`/help`	Affiche la liste des commandes disponibles
`/invite <nick>`	Invite un utilisateur sur le canal où on se trouve
`/join <canal> [clé]`	Permet de rejoindre un canal (protégé éventuellement par une clé). Le canal est créé s'il n'existe pas.
`/list`	Affiche la liste des canaux sur IRC
`/msg [canal|nick] message`	Pour envoyer un message à un utilisateur ou sur un canal (où on est présent ou pas). Les arguments `canal` ou `nick` sont optionnels.
`/names [channel]`	Affiche les utilisateurs connectés à un canal. Si le canal n'est pas spécifié, affiche tous les utilisateurs de tous les canaux.

D'un point de vue utilisateur, un client IRC sera lancé sur la ligne de commande à l'aide du programme `irc.py` de la manière suivante :

```
> python irc.py nickname server_name
```

où `nickname` est le nom du client qui souhaite se connecter sur le serveur IRC `server_name`. Par exemple, cette commande peut lancer une interface graphique (ici réalisée en *tkinter*) semblable à la capture d'écran suivante.



Cette interface est composée de deux zones :
— Un zone pour afficher les messages des autres clients ;
— Un zone de saisie pour composer les messages et exécuter des commandes.

Les serveurs IRC seront lancés avec la commande suivante :

```
> python server.py server_name [servers]
```

où `server_name` représente le nom du nouveau serveur IRC et `servers` est une liste (éventuellement vide) de serveurs auxquels le nouveau serveur doit se connecter.

Important. Pour simplifier l'architecture de ce projet, les noms des serveurs seront simplement des *numéros de port*. Tous les serveurs seront donc lancés sur l'adresse locale de vos machines (`localhost`).

Première version du projet : Dans cette première version, on utilisera un *unique* serveur IRC auquel tous les utilisateurs vont se connecter. En interne, ce serveur doit donc gérer une liste de clients et une liste de canaux. On implémentera toutes les commandes décrites ci-dessus.

Deuxième version du projet : Dans cette version, il s'agit de connecter plusieurs serveurs entre eux, de réaliser l'acheminement des messages et des informations (liste des canaux, des utilisateurs présents dans ces canaux, etc.) entre les serveurs.