**Université Côte d'Azur**
**Faculté des Sciences - Département d'Informatique**

# Licence d'Informatique L2
# Introduction aux Systèmes et Réseaux

## TP n° 6 : Client-serveur, Réseaux, `select`

Ce TP est dédié à la conception d'une architecture client-serveur efficace, utilisée pour la création d'un serveur de *chat*. Il est impératif d'avoir bien maîtrisé les concepts vus lors du TP précédent.

## 1 Réseaux/sockets

Implémenter les exercices de la planche de TD réseaux et les tester. Pour ce faire, utilisez le client générique `telnet`. Usage : `telnet hôte port`.

- `hôte` peut être soit l'adresse IP du serveur, soit `localhost` si celui-ci est sur la même machine que votre client,
- `port` est le numéro de port auquel est rattaché votre serveur.

Pour quitter telnet : d'abord, `CTRL-]`, puis `quit`.

## 2 Création d'un mini-chat

Implémentez un mini-chat fonctionnant en réseau. Vous pouvez partir du client python générique synchrone suivant, que vous devrez peut-être modifier ultérieurement (avec `select`) pour en faire un client asynchrone :

```
import os, socket, sys

MAXBYTES = 4096

if len(sys.argv) != 3:
    print('Usage:', sys.argv[0], 'hote port')
    sys.exit(1)

HOST = sys.argv[1]
PORT = int(sys.argv[2])

sockaddr = (HOST, PORT)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # IPv4, TCP
s.connect(sockaddr)
print('connected to:', sockaddr)
while True: # Client synchrone !! On alterne écriture vers serveur
            # et lecture depuis serveur. Le serveur doit donc lui aussi alterner
    line = os.read(0, MAXBYTES)
```

```
    if len(line) == 0:
        s.shutdown(socket.SHUT_WR)
        break
    s.send(line)
    data = s.recv(MAXBYTES) # attention, si le serveur n'envoie rien on est bloqué.
    if len(data) == 0:
        break
    os.write(1, data)
s.close()
```

### 2.1

Chaque client doit a minima :

- posséder un identifiant court appelé *pseudo*,
- au démarrage d'un client, il doit afficher en premier lieu un message d'accueil et la liste des commandes supportées ;
- pouvoir connaître la liste des autres clients connectés,
- pouvoir envoyer un message à un client donné *@pseudo*,
- pouvoir envoyer un message à tous *@tous*,
- gérer proprement une déconnexion (CTRL-D au clavier),
- sortir proprement à la réception de CTRL-C.

Note : afin que chaque client connaisse la liste des autres clients connectés, deux possiblités (non exclusives) : 1) à la connexion/déconnexion d'un client, le serveur notifie tous les autres clients ; 2) à tout moment, chaque client peut demander au serveur la liste des clients connectés.

Prenez-soin de faire une application facilement testable par un utilisateur (i.e. sans devoir aller voir dans le code) :

- Les noms de vos programmes doivent être parlants ;
- Le serveur doit afficher un usage clair lors d'une tentative d'exécution erronée. Lors du lancement, il doit afficher clairement son numéro de port et son @IP ;
- Le client doit également afficher son usage lors d'une exécution erronée.

### 2.2

Dans cette partie, il est demandé de faire évoluer le serveur. L'utilisateur gérant le serveur doit pouvoir a minima à tout moment :

- Envoyer un message à tous les clients (`wall`) ;
- Bannir proprement un client particulier (`kick`) ;
- Terminer avec délai (`shutdown n`) qui 1) envoie un message à tous les clients indiquant que le serveur va s'arrêter dans $n$ secondes, puis au bout des $n$ secondes, fermer (proprement) ;
- au démarrage du serveur, il doit afficher en premier lieu la liste des commandes supportées.