TP – Programmation réseau

L'objectif de ce TP est de mettre en œuvre une communication entre deux ordinateurs à travers le réseau.
Lisez bien toutes les informations (notamment la partie « Quelques informations sur le module socket ») avant de commencer à faire le programme.

1) Principe général

Nous avons besoin de transmettre et recevoir des informations au moyen de la carte réseau de l'ordinateur. Pour ce faire, nous allons utiliser les fonctions du module **socket** de python.
Ses fonctions permettent l'ouverture de connecteurs (« sockets » en anglais) qu'on peut assimiler à des micro-programmes faisant l'interface avec la carte réseau. Les sockets permettent d'écouter sur un port et d'envoyer des données par le réseau.
Comme d'autres programmes utilisent la carte réseau, il est nécessaire de spécifier un **port** afin de distinguer les programmes destinataires ou émetteurs des différentes informations.
Pour ce TP nous utiliserons le **port 6000**.

La communication se fait de manière dissymétrique : un des ordinateurs jouera le rôle du **serveur** et un autre celui du **client**. Il existe aussi des moyens de communications symétriques où chaque ordinateur est à la fois client et serveur mais nous ne les utiliserons pas ici.
Le pare-feu en charge de protéger les ordinateurs peut bloquer l'ouverture d'un « socket » serveur (mesure de sécurité pour éviter les actions de programmes malveillants). Le serveur sera donc lancé par le professeur, chaque élève aura à concevoir un programme « client » qui communiquera avec ce serveur.

2) Programme client

Tout d'abord le serveur est lancé (voir le programme fourni `serveur.py`).
Il ouvre un « socket » sur le port 6000 sur la carte réseau et « écoute » jusqu'à l'arrivée d'une information qui lui est destiné. Ensuite, il envoie une réponse ou un accusé de réception à l'ordinateur émetteur de l'information.
Le programme « serveur » n'a donc pas besoin de spécifier d'adresse, puisqu'il se contentera répondre à des ordinateurs dont l'adresse est contenue dans les informations d'encapsulation du message.
A l'aide d'une boucle, le programme « serveur » reprend son écoute après avoir envoyé la réponse.

Le programme « client » fonctionne d'une manière un peu similaire. Il ouvre un « socket » sur le port 6000 de la carte réseau pour « émettre » une information vers un autre ordinateur. Il est donc indispensable de spécifier l'adresse de l'ordinateur destinataire (il s'agit de l'adresse du serveur NSI : nsijoliotcurie.fr).
Le programme client envoie une information puis écoute jusqu'à recevoir une réponse ou un accusé de réception.
Dans un second temps, il s'agira de concevoir une boucle pour pouvoir émettre plusieurs messages.

Détails des programmes à réaliser

Client version 1.0
Le programme devra :
- ouvrir un « socket » client sur le port 6000 en direction de l'adresse du serveur (`nsijoliotcurie.fr` ou `82.165.241.240`)
- envoyer un message unique (une chaîne fixée) vers le serveur
- afficher la réponse reçue
- fermer le « socket client »

Client version 2.0
Le programme devra :
- ouvrir un « socket » client sur le port 6000 en direction de l'adresse IP du serveur
- envoyer un message saisi par l'utilisateur après l'avoir « encapsulé » avec une information indiquant le nom de l'auteur du message (On pourra procéder par concaténation de chaînes en faisant précéder le message par « *[votre nom]* dit : »
- afficher la réponse reçue
- fermer le « socket client »

Client version 3.0
Le programme devra exécuter en boucle la version 2.0 tant que :
- le serveur n'aura pas envoyé son message de fermeture (« EoT »)

ou
- l'utilisateur n'aura pas indiqué qu'il souhaite mettre un terme à la communication

Le programme serveur (voir `serveur.py`) étant conçu d'une manière similaire, il peut servir de modèle.

Quelques informations sur le module socket :

Les informations envoyées via les méthodes `send` et `recv` sont toujours sous forme de flux d'octets (type `bytes`).
On peut créer un type objet `bytes` à partir d'une chaîne en la faisant précéder d'un 'b' minuscule (ex : `b'Python'` correspond aux octets permettant de stocker les lettres de la chaîne 'python').
Pour convertir une chaîne existante en bytes, on utilise la fonction du même nom :
`monObjetBytes = bytes(maChaine, 'utf-8')` l'encodage précisé ici étant utf-8, mais on peut en utiliser un autre (non recommandé).
Pour convertir un objet bytes en chaîne de caractère, on utilise la méthode `decode` :
`chaineConvertie = monObjetBytes.decode('utf-8')` là encore on doit préciser l'encodage et il doit bien sûr être le même que celui utilisé pour créer cet objet bytes.

Les méthodes utiles pour la création du programme client sont :

`socket.setdefaulttimeout(valeur_timeout)` qui permet de fixer un temps limite de réponse et éviter de bloquer le programme client en cas de soucis sur la communication.
`socketClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)` permet de créer le socket client.

`socketClient.connect((IP_SERVEUR, PORT))` permet de connecter le socket à un serveur distant dont on doit donner l'IP et le port. Attention : l'adresse IP et le port sont donnés sous la forme d'un *tuple* (donc entre parenthèses)
`socketClient.send(objet_bytes_à_transmettre)` qui envoie un message sous forme d'objet byte vers le réseau via le socket.
`objetByteRecu = socketClient.recv(4096)` récupère un message qui arrive sur le socket (la valeur 4096 correspond à la taille du buffer[1] de réception)
`socketClient.close()` Ferme le socket (et donc la connexion)

Pour davantage d'information sur le module sockets, on peut directement consulter sa documentation en ligne (en anglais) : https://docs.python.org/3/library/socket.html ou moins complet, mais en français : https://docs.python.org/fr/3/library/socket.html .

[1] Un buffer est une mémoire tampon, c'est-à-dire une mémoire (généralement de taille réduite) où on stocke temporairement des données avant de les traiter.

3) Communication entre deux programmes

Maintenant vous allez vous mettre par deux et concevoir deux programmes capables de communiquer entre eux par le réseau un peu à la manière des programmes serveur et client que nous venons d'utiliser. Choisissez ensemble qui réalise le programme serveur et qui code le client.

Programme serveur :
Le programme doit ouvrir un socket en serveur sur le port 6400 puis choisir un nombre entier secret au hasard entre 1 et 30.
Il écoute les communications et si on lui envoie un nombre entier, il le compare au nombre secret et répond « Trop grand ! » si le nombre qu'il a reçu est plus grand que le nombre secret, « Trop petit ! » s'il est plus faible et « Gagné » s'il est égal. Dans ce dernier cas le programme s'arrête, sinon il attend un nouveau message jusqu'à ce que l'utilisateur distant trouve.

Programme client :
Le programme demande l'adresse IP du serveur, puis ouvre une connexion vers le serveur sur le port 6400. Il demande en boucle quel nombre on choisit et envoi ce nombre au serveur puis affiche la réponse de ce dernier.
Le programme se termine quand le serveur répond « Gagné ».

Pour connaître l'adresse IP de votre poste, rappelez-vous de ce que vous avez vu dans le cours (commande `ipconfig` et site http://www.mon-ip.com/). Si vous voulez communiquer entre deux ordinateurs du même réseau local (au lycée), il faut utiliser l'adresse interne (donnée par `ipconfig`). Si vous communiquez avec un ordinateur d'un autre réseau, il faut utiliser son adresse externe. Enfin un ordinateur du réseau local du lycée ne peut pas faire serveur pour un ordinateur en dehors de son réseau (sur internet) à cause des sécurités mises en place au niveau du routeur.

Une fois que vos programmes fonctionnent bien, Essayez de les améliorer :

Serveur v2 : Une fois que le nombre secret est deviné, le serveur en tire un nouveau et recommence jusqu'à ce qu'il reçoive « `EoT` » du client. Lorsqu'on a deviné, le serveur annonce le nombre de coup nécessaires.

Serveur v3 : Le serveur peut gérer plusieurs clients simultanément et renvoyer à chaque client sa réponse et compter séparément le nombre de coups pour chaque client. Il ne s'arrête qu'une fois que tous les clients ont trouvé et à ce moment là il leur envoi le nombre de coup du meilleur client avant de relancer une partie.

Client v2 : Le client vérifie le joueur ne fait pas de propositions absurdes (ex : demander un nombre plus grand que la fois précédente alors que le serveur a répondu que c'était plus petit).

Client v3 : Le client propose un mode de jeu automatique où c'est l'ordinateur qui élabore la proposition de nombre et l'envoi au serveur jusqu'à trouver le bon nombre. On fera une pause de 200 ms entre chaque proposition pour ne pas surcharger le serveur.