TP n°1 *Socket, UDP, Client/Server, Broadcasting*

Dans ce TP, vous allez découvrir comment des applications peuvent communiquer entre elles au travers du réseau. Tous les TPs doivent être versionnés. Donc commencez par créer un projet pour les TPs réseau ; puis un sous répertoire pour ce premier TP :

```
git init tps-reseau
cd tps-reseau
mkdir TP1
cd TP1
```

Exercice 1. Serveur de date/heure par UDP.

PAS DE COPIER-COLLER ! Vous devez retaper le code ci-dessous en suivant les explications de votre responsable de TP.

fichier `date-server.py`

```
#! /usr/bin/python3

import socket
import datetime

BUFSIZE = 1024

def server(port):
    sock = socket.socket(type=socket.SOCK_DGRAM)
    sock.bind(("0.0.0.0", port))
    while True:
        data, addr = sock.recvfrom(BUFSIZE)
        data = datetime.datetime.now()
        data = "%s\n" % data
        sock.sendto(data.encode(), addr)

server(5555)
```

N'oubliez pas de faire un `git add` et un `git commit` pour enregistrer ce programme dans votre projet.

Exercice 2. Utilisation de `netcat`.

Vous allez pouvoir tester ce serveur, en vous y connectant grâce à l'utilitaire en ligne de commande `netcat` :

```
netcat -4 -u localhost 5555
```

Rien ne se passe ! En effet, il faut que provoquiez l'envoi d'un message au serveur pour que celui-ci vous réponde. Si vous entrez une ligne de texte, `netcat` l'encapsulera dans un message et enverra celui-ci au serveur. Il vous suffit donc d'appuyer sur la touche Entrée. Si vous appuyez plusieurs fois, chaque fois une message sera envoyé et le serveur vous enverra une réponse.

Pour arreter `netcat`, utilisez `Control-C`.

Exercice 3. Client pour le service de date/heure.

PAS DE COPIER-COLLER ! Vous devez retaper le code ci-dessous en suivant les explications de votre responsable de TP.

fichier `date-client.py`

```
#! /usr/bin/python3

import socket

BUFSIZE = 1024

def client(host, port):
    sock = socket.socket(type=socket.SOCK_DGRAM)
    addr = (host, port)
    sock.sendto(b"", addr)
    data = sock.recv(BUFSIZE)
    print(data.decode(), end="")

client("localhost", 5555)
```

Testez ce client en l'invocant à la ligne de commande. N'oubliez pas de committer.

Exercice 4. Passage d'argument au client sur la ligne de commande.

Le module `sys` (voir `https://docs.python.org/3/library/sys.html`) exporte la variable `argv` qui contient une liste de chaînes : ce sont les éléments de la ligne de commande. Servez-vous en pour permettre à l'utilisateur de donner sur la ligne de commande un argument indiquant le nom ou l'IP de la machine où contacter le serveur. Testez cette modification en interrogeant le serveur de votre voisin.

Exercice 5. Serveur/client extensibles.

Pour l'instant le serveur ne sait fournir que la date. Vous allez en créer un nouveau qui saura répondre à différentes requètes. Pour cela, vous allez dupliquer les codes que vous avez :

```
cp date-server.py ext-server.py
cp date-client.py ext-client.py
```

et vous modifirez ces nouveaux fichiers (faites les add/commit qui s'imposent). Quelle autre information pourrait-on vouloir publier ? Par exemple l'identifiant de la personne qui fait tourner le serveur. Voici comment obtenir cette info :

```
$ python3
Python 3.3.5 (default, Mar 29 2014, 16:54:39)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> import os
>>> os.environ["USER"]
'denys'
>>>
```

Voici comment le client envoit un message au serveur :

```
sock.sendto(b"", addr)
```

Notez que le message est vide. Pour permettre au client de faire différentes requètes, il suffirait que le message contienne un mot indiquant l'information voulue :

```
sock.sendto(b"date", addr)
```

ou bien :

```
sock.sendto(b"user", addr)
```

De son côté le serveur doit examiner le mot reçu dans le message du client, et doit renvoyer l'information correspondante.

Il faudrait que l'utilisateur puisse invoquer le client comme ceci :

```
./ext-client.py localhost date
./ext-client.py localhost user
./ext-client.py IP-DU-VOISIN date
./ext-client.py IP-DU-VOISIN user
```

où `IP-DU-VOISIN` serait remplacé par l'adresse IP de votre voisin, pour interroger son serveur.

Exercice 6. Broadcasting server.

Il est également possible de broadcaster des messages de sorte que tous les clients d'un réseau local puissent les recevoir.

fichier bcast-server.py

```
#! /usr/bin/python3

import socket
import datetime
import time

def server(port):
    sock = socket.socket(type=socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    addr = ("255.255.255.255", port)
    while True:
        data = str(datetime.datetime.now())
        sock.sendto(data.encode(), addr)
        time.sleep(1.0)

server(6666)
```

fichier bcast-client.py

```
#! /usr/bin/python3

import socket

def client(port):
    sock = socket.socket(type=socket.SOCK_DGRAM)
    addr = ("255.255.255.255", port)
    sock.bind(addr)
    while True:
        data, saddr = sock.recvfrom(128)
        print(saddr, data.decode())

client(6666)
```

Notez que le client affiche l'addresse du serveur dont provient le message. Ceci vous permettra de constater que vous recevez également les messages provenant des serveurs de vos collègues.