**Projet Serveur - Client**

**Description :**

Il s’agit de réaliser une relation Serveur – Client en utilisant le switch de la salle E314. Le but de ce projet est de réaliser un « tchat » à deux.



**Le principe d’une relation Client – Serveur :**

Réponse

Requête

Serveur

(Fournit un service)

Client

(Demande un service)

Le schéma client-serveur est un concept fondamental pour la programmation d’applications réparties (communication entre deux processus s’exécutant sur des machines distinctes, reliées par un réseau).

Pour le client, un service est souvent désigné par un nom symbolique. Ce nom doit être converti en une adresse interprétable par les protocoles du réseau. La conversion d’un nom symbolique (par exemple www.google.com) en une adresse IP (216.239.39.99) est à la charge du service DNS.

En fait, l’adresse IP du serveur ne suffit pas, car le serveur (machine physique) peut comporter différents services ; il faut préciser le service demandé au moyen d’un numéro de port, qui permet d’atteindre un processus particulier sur la machine serveur. Un numéro de port comprend 16 bits (0 à 65 535) et est associé à un protocole de transport donné (le port TCP numéro x et le port UDP numéro x désignent des objets distincts).

Les numéros de 0 à 1 023 sont réservés, par convention, à des services spécifiques.

**Port 50000**

Adresse IP :

192.168.1.11

Connexion TCP

Serveur

(Fournit un service)

Client

(Demande un service)

Pour programmer une application client-serveur, il est commode d’utiliser les sockets. Les sockets fournissent une interface qui permet d’utiliser facilement les protocoles de transport tels que TCP et UDP.

Un socket est simplement un moyen de désigner l’extrémité d’un canal de communication bidirectionnel, côté client ou serveur, en l’associant à un port.

192.168.1.11 : 50000

192.168.1.12 : 45000

Serveur

(Fournit un service)

Client

(Demande un service)

Socket

Serveur

Socket

Client

**Sockets TCP côté client (1) :**

On procède en plusieurs étapes : on suppose que l’on connaît l’adresse IP d’un serveur et le numéro de port (TCP) d’un socket serveur. **Le processus serveur est en attente sur ce port**.

On utilise la bibliothèque **socket** intégré de base à Python qu’il faut importer.

**Etape 1 :** *Création du* *socket client*.

Il faut faire appel au constructeur ***socket***. Dans le cas d'une connexion TCP, il prend les deux paramètres suivants, dans l'ordre :

**socket.AF\_INET** : la famille d'adresses, ici ce sont des adresses Internet IPV4 sinon **AF\_INET6** pour IPV6 ;

**socket.SOCK\_STREAM** : le type du socket, SOCK\_STREAM pour le protocole TCP.

Socket serveur

Socket client

Le serveur est en attente sur le socket (accept)

Client

IP + port

Serveur

On utilise la ligne de commande :

connexion\_avec\_serveur = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

**Etape 2 :** *Etablir une connexion entre le socket client et le serveur*.

Il faut connaitre les deux informations principales pour établir la connexion entre le client et le serveur : l’adresse IP (nommée HOST de type ***str***) du serveur et le port de ce dernier (nommé PORT de type ***int***). Les deux sont des variables fixes et donc d’après le PEP8, ces variables sont en lettres majuscules.

HOST = "192.168.1.**12**" # le dernier nombre celui du serveur (de ton binôme) !

PORT = 50000 # choisit par le serveur.

On utilise ensuite la méthode ***connect***. Elle prend en paramètre un tuple composé de l’adresse IP et du port du serveur. La ligne de commande est la suivante :

connexion\_avec\_serveur.***connect***((HOST, PORT))

Socket serveur

Socket client

Connexion établie

Client

Port

Serveur

***connect***

La méthode ***connect*** envoie une demande de connexion vers le socket serveur : une exception **OSError** est levée en cas d’échec de la tentative de connexion …

Maintenant votre serveur et votre client sont connectés ! Si vous retournez dans la console Python abritant le serveur, vous pouvez constater que la méthode ***accept*** ne bloque plus, puisqu'elle vient d'accepter la connexion demandée par le client.

**Etape 3 :** *Faire communiquer les sockets*.

Pour faire communiquer les sockets, on utilise les méthodes ***send*** pour envoyer et ***recv*** pour recevoir. Les informations que vous transmettrez doivent être des chaînes de ***bytes*** et non pas de type ***str*** !

La méthode ***recv*** prend en paramètre le nombre de caractères à lire. Généralement on lui passe la valeur **1024**. Si le message est plus grand que 1024 caractères, on récupérera le reste après.

Pour régler ce souci, on peut utiliser les méthodes ***encode*** et ***decode*** qui pour l’une encode une variable de type ***str*** en ***bytes*** et pour l’autre décode une variable de type ***bytes*** en ***str***.

**Envoi d’un message au serveur :**

message = message.***encode*()** # conversion de la variable de type str en bytes.

connexion\_avec\_serveur.***send***(message) # envoi du message au serveur.

**Réception d’un message du serveur :**

msg\_reçu = connexion\_avec\_serveur.***recv***(1024) # réception du message envoyé par le serveur de type bytes.

msg\_reçu = msg\_reçu.***decode*()** # conversion de la variable de type bytes en str.

**Etape 4 :** *Fermer la connexion*.

Pour fermer la connexion, il faut appeler la méthode ***close*** de notre socket.

connexion\_avec\_serveur.***close***()

Les étapes précédentes réalisent un serveur en mode itératif : un seul client est servi à la fois.

**Résumé :**

Connexion initiale

**Client**

**Serveur**

socket()

connect()

send()

revc()

close()

socket()

bind()

listen()

accept()

revc()

send()

close()

Requête

Réponse

**Boucle itérative client-serveur**

A toi de jouer :

* Votre programme doit avoir :
  + Une fonction ***envoyer*** **(***texte***)** dont le rôle est d’envoyer un message au serveur et qui prend en paramètre le texte à envoyer de type ***str***.
  + Une fonction ***recevoir*** **()** dont le rôle est de recevoir un message du serveur. Elle retourne ce message en une chaîne de caractères de type ***str***.
  + Une fonction ***fin\_connexion*** **()** dont le rôle est de fermer la connexion avec le serveur.
  + Le programme principal doit gérer :
    - La création du socket client ;
    - Les variables HOST et PORT pour établir la connexion avec le serveur ;
    - La boucle itérative (indéterministe) pour l’envoi et la réception des messages avec le serveur ;
    - La fermeture de la connexion.

**Améliorations possibles :**

* …