**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 serveur (1) :**

Un serveur en mode connecté doit attendre une nouvelle demande de connexion de la part d’un client, puis traiter la (ou les requêtes) envoyée(s) sur cette connexion par le client.

Les fonctions d’attente et de traitement sont séparées, pour permettre au serveur d’attendre de nouvelles demandes de connexion pendant qu’il traite des requêtes en cours.

Socket serveur

En attente

Serveur

Le socket serveur est associé à un port connu des clients : par exemple 50000

Socket serveur

Socket de communication

Demande de connexion

Serveur

Création

Le socket de communication est associé au même numéro de port que le socket serveur, mais avec un descripteur de (pseudo) fichier différent.

On procède en plusieurs étapes : on suppose que l’on connaît l’adresse IP du serveur et le numéro de port (TCP) du socket serveur.

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

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

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

Serveur

On utilise la ligne de commande :

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

**Etape 2 :** *Associer le socket à une adresse IP et un numéro de port TCP locaux.*

Il faut utiliser les deux informations principales pour configurer le socket du serveur : son adresse IP (nommée HOST de type ***str***) et son port (nommé PORT de type ***int***) sur lequel le serveur doit écourter les demandes de connexion. 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 est donné en fonction de ta place dans la salle !

PORT = 50000 # choisir un nombre supérieur à 1023.

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

connexion\_principale.***bind***((HOST, PORT))

Socket serveur

Port

Serveur

**Etape 3 :** *Indiquer que le socket est serveur*.

Un socket serveur sera en attente de demande(s) de connexion. Il lui faut gérer une « file » d’attente des demandes. On utilise la méthode ***listen*** sur le socket serveur

Socket serveur

File

Serveur

MAX\_CONNEXION = 5 # par exemple …

connexion\_principale.***listen***(MAX\_CONNEXION)

Après un appel réussi à ***listen***, le système d’exploitation de la machine serveur peut **commencer à recevoir des demandes de connexion**. Si une demande de connexion arrive pendant qu’une autre est en cours de traitement, elle est placée dans une file d’attente. Si une demande arrive alors que la file est pleine, elle est rejetée.

**Etape 4 :** *Prise de connaissance des nouvelles connexion – Obtention d’un canal de dialogue côté serveur*.

On utilise la méthode ***accept***. Elle est **bloquante** si la file est vide. En retour de cette méthode, on obtient un tuple composé en premier d’une variable qui contient une référence vers un objet socket permettant de dialoguer avec un client. Le second élément du tuple contient des informations sur le client qui demande la connexion.

connexion\_avec\_client

connexion\_principale

Socket serveur

Socket de dialogue

Demande de connexion

File

Serveur

Création

(connexion\_avec\_client, client\_info) = connexion\_principale.***accept***()

Le serveur peut maintenant communiquer avec le client.

**Etape 5 :** *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 client :**

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

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

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

msg\_reçu = connexion\_avec\_client.***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 5 :** *Fermer les connexions*.

Pour fermer les connexions, il faut appeler la méthode ***close*** sur le socket de dialogue et sur le socket du serveur.

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

connexion\_principale.***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 client et qui prend en paramètre le texte à envoyer de type ***str***.
  + Une fonction ***recevoir*** **()** dont le rôle est de recevoir un message du client. 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 socket serveur et le socket de dialogue.
  + Le programme principal doit gérer :
    - La création du socket serveur ;
    - Associer le socket aux paramètres HOST et PORT ;
    - Déclarer avec ***listen*** que le socket est serveur. Le nombre maximum de connexion accepté est pour le moment fixé à 1 ;
    - Prendre connaissance de la nouvelle connexion avec ***accept*** ;
    - La boucle itérative (indéterministe) pour l’envoi et la réception des messages avec le client ;
    - La fermeture de la connexion.

**Améliorations possibles :**

* …