TD/TP ASR5 n°9/10 : les communications réseaux

Notions abordées :

- Langage Python
- Les sockets
- Connexion Client-serveur

I – Le langage Python – les références

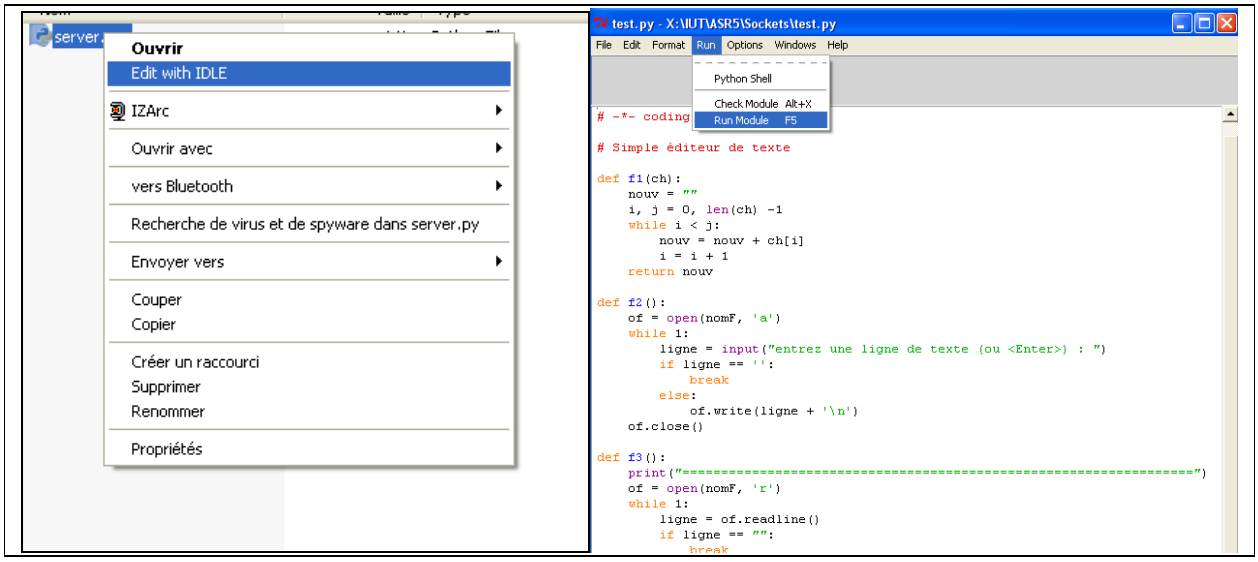
Les sockets en python : http://docs.python.org/py3k/library/socket.html#

Un très bon livre de référence de Gérard Swinnen : http://www.inforef.be/swi/python.htm. Le chapitre 18 est consacré aux communications réseaux.

Initiation à Python 3 (disponible dans S2T\ASR5\TP\TP9-10\Python3v1-1.pdf/) : http://www.afpy.org/Members/bcordeau/Python3v1-1.pdf/download

II – Premiers pas en python

Vous utiliserez l'éditeur IDLE pour écrire vos programmes en python :



Vous ouvrez pour cela un fichier python (avec l'extension .py) avec l'éditeur IDLE, puis lancez l'exécution avec Run>Run Module ou à l'aide de la touche F5 ou encore en double-cliquant sur le fichier `test.py` dans votre explorateur windows.

Ouvrez le fichier `test.py` et lancez son exécution. Ce fichier vous aidera à comprendre la syntaxe python.

1. Décrivez le fonctionnement du programme et donnez une définition pour chaque fonction disponible dans ce fichier. Cette définition sera placée en commentaire au début de

chaque fonction.

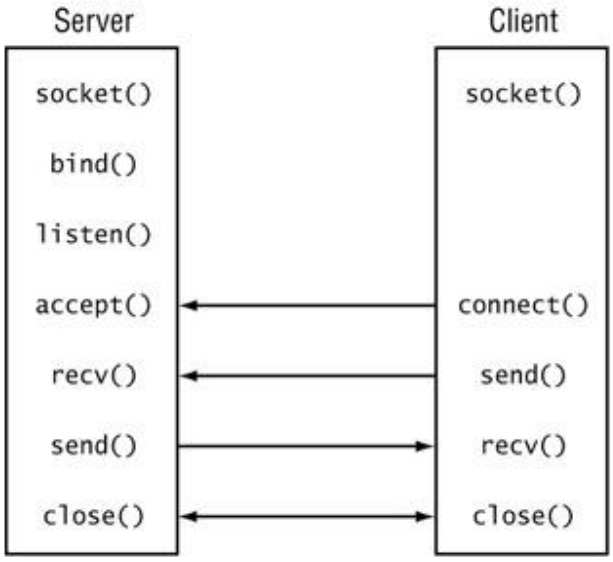
2. Remplacez la ligne `print(f1(ligne))` dans la fonctions `f3`, par :
   - `print(ligne)`, puis par
   - `print(ligne,end='')`

   Testez ces 2 possibilités avec une consultation de fichier (tapez `c` après avoir rentré un nom de fichier).
3. Ecrivez un premier programme en python (à partir de ce que vous avez vu à la première question) qui affiche 'Hello World' à l'écran dans un premier temps puis dans un fichier `test.txt` dans un second temps.
4. Réalisez un programme qui affiche la suite des symboles suivants (vous pourrez faire varier dans votre programme le nombre d'étoiles de la dernière ligne) :

```
*
**
***
****
*****
******
*******
```

III - La programmation Client-serveur à l'aide des sockets

Vous allez créer tout d'abord un fichier `serveur.py` qui effectuera dans l'ordre les fonctions détaillés dans la figure ci-dessous.



1. Donnez en quelques lignes les fonctionnalités principales de chaque fonction, en détaillant

pour chacune de ces fonctions les paramètres utilisés.

Nous cherchons à réaliser dans un premier temps une application client-serveur où le client réussit à se connecter au serveur et le serveur affiche les informations de connexion du client (`adresse IP et port`).

2. Codez la partie serveur (`socket, bind, listen, accept, close`) et la partie client (`socket, connect, close`) en python. Vous devrez indiquer en entête de votre fichier `import socket`.

Vous aurez sur le serveur un affichage de ce genre :

```
>>>
Le serveur est prêt à recevoir des connexions....
Un client s'est connecté :  ('138.96.41.88', 3239)
```

IV – Un serveur web simple en quelques lignes

Maintenant dès que le client a réussi à se connecter, le client et le serveur vont échanger des informations via les fonctions `send()` et `recv()`.

1. Le client envoie la phrase `'Bonjour je souhaite me connecter'` dès qu'il se connecte et le serveur affiche ce qu'il reçoit du client à l'écran.
2. Maintenant, le serveur renvoie le contenu du fichier `index.html` (et le client affiche ce qu'il a reçu de la part du serveur). Il faudra pour cela copier le fichier dans un buffer qui sera envoyé au client.
3. Si vous vous connectez au serveur via un navigateur, la page `index.html` doit s'afficher dans le navigateur. Vous remarquerez aussi que le serveur affiche la requête `GET` qu'il a reçue.
4. La capture wireshark `socket.cap` a été réalisée lors de cette connexion. Vous pouvez cliquer sur le paquet n°11 contenant une requête GET, et choisir 'Decode As' Transport Layer as http. Tous les paquets contenant des requêtes http seront colorés en vert.
5. Vous pouvez aussi vous connecter à ce serveur via telnet.

V – Un serveur web plus évolué

Votre serveur doit recevoir des requêtes `http` de la part de clients qui se connectent et doit répondre à ces requêtes. Lors de la première connexion, le client voit s'afficher une page contenant un formulaire (`index.html`), puis une fois le formulaire rempli, et les données envoyées vérifiées par le serveur, la page `home.html` s'affichera. Pour réaliser cela, les étapes suivantes vont être effectuées :

1. Le serveur doit ouvrir un socket qui s'attache à l'adresse de la machine locale sur un port **donné en paramètre**.
2. Le serveur attend les connexions des clients.

3. Dès qu'il y a connexion, le serveur accepte la connexion, et garde une trace de cette connexion dans un fichier de `log`.
4. Le serveur lit la requête `http` puis répond :
    - S'il s'agit d'une requête `GET`, le serveur regarde quelle page le client demande et renvoie la page adéquate au client (`index.html` si la requête est de la forme `GET / HTTP/1.1`). Si l'objet de la requête n'est pas trouvé, le message de réponse contient la ligne de statut : `HTTP/1.1 404 Not Found` et un corps de réponse vide.
    - S'il s'agit d'une requête `POST`, le serveur vérifie les informations qu'il a reçues et affiche à l'écran chacun des paramètres remplis par le client. Puis, le serveur renvoie la page adéquate au client
5. A la fin de la gestion de la requête, le serveur retourne en état attente de connexions (étape 2).

Vous respecterez bien les consignes suivantes :

- Vous apporterez un soin tout particulier à la **gestion des erreurs** en renvoyant des messages explicites d'erreurs au client et au serveur directement dans le terminal dans lequel il a été lancé.
- Le serveur enregistrera les informations de connexion des clients dans un fichier de log, avec l'heure exacte de la connexion des clients.
- Votre code sera **commenté**
- Vous gérerez aussi les requêtes pour les images favicon.ico (dont la génération peut se faire sur ce site http://www.favicon.cc) qui s'affichent dans la barre de navigation à côté l'url. La requête associée est de la forme suivante :

```
GET /favicon.ico HTTP/1.1
Host: 138.96.199.71:50002
User-Agent: Mozilla/5.0 (X11; U; Linux i686; fr; rv:1.9.2.16)
 Gecko/20110322 Fedora/3.6.16-1.fc14 Firefox/3.6.16
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-us,en;q=0.7,fr;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
```

- Votre serveur web gèrera une (ou plusieurs) commande réseau que nous avons vu en TP, telle que `ping`, `traceroute`, `nmap`, `whois`… Le fichier `test-tracert.py` vous donne un exemple d'utilisation de traceroute. Vous êtes libres d'implémenter la commande réseau que vous souhaitez.
- Les informations du client (@IP, système d'exploitation, navigateur web…) seront affichées dans la page `index.html` (elles seront lues à partir de la requête `GET`)
- La page `index.html` sera donc un formulaire qui demandera les informations dont vous avez besoin pour lancez la commande. Par exemple une adresse IP pour faire un `traceroute`.
- La page `home.html` présentera le résultat donné par cette commande.

Vous rendrez pour le 27 mai 2011, avant 13h00 (délai de rigueur), seul, par binôme, mais pas par trinôme :

- Votre code **COMMENTÉ** du serveur en python
- Un rapport synthétique de 5 à 10 pages décrivant les fonctionnalités implémentées dans votre serveur