Réseau

GCC ! – Prologin

2020

Introduction

Lorsque deux ordinateurs veulent communiquer sur un réseau, il faut que leur système d'exploitation respectif sache à quel programme transmettre les informations échangées. Les *sockets* sont un moyen mis à disposition par le système d'exploitation pour que les programmes puissent indiquer qu'ils veulent envoyer ou recevoir ces informations.

Un réseau informatique fonctionne exactement comme le réseau de distribution du courrier. Pour envoyer une lettre à quelqu'un, on indique son adresse postale et son nom (afin de pouvoir distinguer les personnes habitant au même endroit). Si on suit cette analogie, l'ordinateur serait la maison, son adresse postale serait son *adresse IP* et le nom du destinataire serait un moyen d'identifier le programme qui doit recevoir les informations : *le numéro de port*.

Afin de se comprendre, il est important de se mettre d'accord au préalable sur un *protocole*. Un protocole est une convention qui permet à des entités distinctes de pouvoir communiquer. En lisant ce texte, vous savez qu'il faut le parcourir de haut en bas et de la gauche vers la droite. Vous connaissez les symboles qui sont utilisés pour former les mots et la signification associée à chaque mot de ce texte. De la même façon, pour que deux programmes puissent communiquer, il faut qu'ils aient été programmés de façon à interpréter de la même façon les informations qu'ils échangent.

Pour ce TP nous allons utiliser un protocole de transport qui se nomme TCP et qui est le protocole généralement utilisé sur internet. Il permet d'avoir des communications fiables : il retransmet automatiquement les données perdues et s'arrange pour qu'elles soient reçues dans l'ordre dans lequel elles ont été envoyées. C'est un protocole dit *orienté connexion*, c'est à dire qu'avant de pouvoir échanger des données, il faut établir une connexion avec son interlocuteur. Avec TCP, un programme fait office de *serveur*, et l'autre fait office de *client*. C'est le client qui initie la connexion vers le serveur. Il aura pour cela besoin de l'adresse de l'ordinateur qui exécute le serveur ainsi que le numéro de port que le serveur s'est choisi [1].

Le but de ce TP est de vous guider dans l'élaboration d'un client de messagerie instantanée.

1. Contrairement aux êtres humains, les programmes choisissent eux-même le « nom » qu'il faut utiliser pour les contacter. Lors de son initialisation, le serveur va choisir le numéro de port qui doit être utilisé pour le contacter.

Manipulation des sockets

Les éléments suivants résument les opérations de manipulation de socket. Ils sont expliqués plus en détail dans la suite de ce TP.

Permet l'utilisation des sockets, à n'utiliser qu'une seule fois, tout au début du code source :

```
1 import socket
```

Initialise une nouvelle socket utilisant les protocoles TCP et IP :

```
1 tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
```

Initie une connexion avec le serveur situé à l'adresse 1.2.3.4 sur le port 4242 :

```
1 tcpsock.connect(("1.2.3.4", 4242))
```

Crée un fichier associé à la socket. Ce fichier ne pourra servir que pour des lectures :

```
1 rfile = tcpsock.makefile("r", encoding="utf-8")
```

Lit une ligne de texte sur la socket :

```
1 texte = rfile.readline()
```

Crée un fichier associé à la socket. Ce fichier ne pourra servir que pour des écritures :

```
1 wfile = tcpsock.makefile("w", encoding="utf-8")
```

Envoie au serveur la chaîne de caractères « coucou » suivie d'un saut de ligne :

```
1 wfile.write("coucou\n")
2 wfile.flush()
```

Termine la connexion et libère les ressources allouées pour la socket :

```
1 rfile.close()
2 wfile.close()
3 tcpsock.shutdown(socket.SHUT_RDWR)
4 tcpsock.close()
```

Lecture sur une socket

Dans un premier temps, nous allons créer une socket, initier une connexion vers un serveur, afficher ce qu'il nous envoie et terminer le programme.

Afin d'utiliser les sockets, nous allons faire appel à des fonctions qui ne sont pas disponibles de base dans le langage (contrairement à la fonction `print()` par exemple). Il faut donc indiquer explicitement que l'on veut avoir accès à ces fonctions. On utilise pour cela la directive « `import` ». Toutes les fonctions et constantes ajoutées par cette directive commenceront par « `socket.` ».

```
1 import socket
```

On va ensuite créer une socket utilisant les protocoles TCP et IP :

```
1 tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
```

AF_INET correspond au protocole IPv4 et SOCK_STREAM indique que l'on veut utiliser un protocole orienté connexion en mode flux (on peut envoyer des données de façon continue), ça correspond bien aux caractéristiques du protocole TCP.

```
1 tcpsock.connect(("<ADRESSE>", 2000))
```

On utilise son adresse et le port 2000.

Sous UNIX, une entité avec laquelle on peut écrire et lire des informations est un fichier (indépendamment du fait qu'il soit stocké sur le disque dur), on peut donc récupérer un fichier lié à la socket afin de pouvoir lire et écrire dessus.

```
1 rfile = tcpsock.makefile("r", encoding="utf-8")
```

Le caractère “r” indique qu'on veut un fichier sur lequel on puisse lire. Le second paramètre est lié à la façon dont les chaînes de caractères sont envoyées sur le réseau.

On peut maintenant lire une ligne de texte sur ce fichier et l'afficher :

```
1 print(rfile.readline())
```

Une fois que c'est fait, on peut fermer la connexion et libérer les ressources allouées pour la socket :

```
1 rfile.close()
2 tcpsock.shutdown(socket.SHUT_RDWR) # Termine la connexion
3 tcpsock.close()
```

Exercice 1 À partir des explications ci-dessus, écrivez un programme qui se connecte à l'adresse inscrite au tableau, se connecte sur le port 2000, affiche une ligne de texte envoyée par le serveur et termine la connexion.

Exercice 2 Adaptez votre programme pour qu'il affiche indéfiniment chaque ligne de texte envoyée par le serveur. On peut faire Ctrl+C pour quitter le programme (pratique si vous n'avez pas le temps d'attendre indéfiniment !)

NB : conservez ce programme dans un fichier appelé client_reception.py.

Exercice 3 Est-il possible de modifier le programme pour qu'il se termine lorsque le serveur a fini d'envoyer les données ? Essayez en vous connectant sur la même adresse, mais au port 2001. Ce serveur envoie un nombre aléatoire de messages.

Écriture sur une socket

Exercice 4 Écrivez un programme qui se connecte au serveur sur le port 4240, qui demande un pseudonyme à l'utilisateur (avec la fonction input()) et qui l'envoie au serveur en utilisant la fonction write sur le fichier lié à la socket. Le pseudo doit être suivi d'un caractère « retour à la ligne » :

```
1 wfile.write(pseudo + "\n")
2 wfile.flush()
```

Bonus : Pourquoi ce caractère est-il nécessaire ?

Exercice 5 Adaptez votre programme pour qu'après avoir envoyé le pseudonyme au serveur, il demande en continu à l'utilisateur de rentrer une ligne de texte. Chaque ligne de texte est immédiatement envoyée au serveur.

NB : conservez ce programme dans un fichier appelé `client_envoi.py`.

Clavardons !

Ouvrez deux terminaux, dans le premier, lancez votre programme `client_reception.py` sur le serveur en utilisant le port 4241. Dans le second, lancez votre programme `client_envoi.py` en utilisant le port 4242.