Programmation répartie. Module 4102C
TP 3 : communications entre machines par le biais de sockets.
année universitaire 2017-2018

Samuel Delepoulle

19 février 2018

Présentation des sockets [1]

Le mécanisme des sockets (littéralement "prise") a été introduit dans le système UNIX BSD (*Berkeley Software Distribution*). C'est un système de communication IPC (*Inter Process Communication*). Ce mécanisme permet donc a des processus de communiquer via un réseau TCP/IP. Les sockets peuvent être implémentées dans de nombreux langages. En C, par exemple, les outils (structures et fonctions) pour les utiliser sont regroupées dans la bibliothèque `<sys/socket.h>`.

Les sockets ont deux modes
— connecté (protocole TCP).
— non connecté (protocole UDP).

Les sockets en Java

Les classes java qui permettent de gérer les sockets sont :
— `Socket` pour le protocole TCP.
— `DatagramSocket` pour le protocole UDP.

1. source Wikipedia

Sockets TCP

Il existe un constructeur :

— `Socket(String host, int port)` avec `host` qui représente l'adresse (logique ou symbolique) de la machine hôte et `port` le port utilisé pour la communication.

Lors de sa création, la socket est implicitement connectée.

On peut alors utiliser les méthodes

— `OutputStream getOuputStream()` qui renvoie un flux de sortie pour la socket.

— `IntputStream getIntputStream()` qui renvoie un flux d'entrée pour la socket.

— `close()` qui permet de fermer la socket.

Comment utiliser les flux d'entrée et de sortie ? L'utilisation des entrées-sorties repose sur plusieurs classes regroupées en fonction des données qu'elles peuvent traiter (voir cours sur les entrées-sorties en général).

Pour simplifier, dans le cas ou on souhaite lire et écrire des caractères, on peut utiliser les méthodes suivantes :

écriture de caractères sur une socket

On construit un objet `OutputStream` et on fabrique un objet d'écriture dessus (classe `PrintWriter`).

```
Socket s = new Socket(host,port);
PrintWriter pw = new PrintWriter(s.getOutputStream());
```

On peut ensuite écrire des chaînes de caractères sur l'objet `pw` en utilisant les méthodes `print(String str)` et `println(String str)`.

Attention : la lecture et l'écriture par l'intermédiaire des réseaux utilise généralement des mémoires tampon (ou *buffer*) afin d'optimiser les échanges. L'utilisation des méthodes d'écriture n'est donc pas une garantie que les informations sont bien transmises. Pour cela, il faut demander que le buffer soit vidé, en utilisant la méthode `flush()`.

lecture de caractères sur une socket

Pour l'écriture, on crée un objet `InputStream` à partir duquel on construit un objet `InputStreamReader(is)`. Cet objet permettra la lecture de caractères uniques (ou de tableaux de caractères). Comme dans la pratique on

préfère généralement lire des chaînes de caractères, il faudra utiliser un objet `BufferReader` qui construit un tampon de lecture sur lequel on pourra lire des chaînes de caractères :

```
Socket s = new Socket(host,port);
BufferedReader bf = new BufferedReader(
        new InputStreamReader(s.getInputStream()));
```

On appelle ensuite la méthode `readLine()` sur l'objet `bf` pour lire une "ligne".

remarque 1 la même socket peut servir pour la lecture et l'écriture.

remarque 2 On peut également utiliser un objet de la classe Scanner pour réaliser les opérations de lecture sur la Socket :

```
Socket s = new Socket(host,port);
Scanner sc = new Scanner(s.getInputStream());
```

Les lectures/écriture sur la socket se font alors par les même méthodes que sur la console : print, println pour l'écriture, nextLine pour la lecture.

Sockets serveur et socket client

La création de socket "client" suit la syntaxe expliquée ci-dessus : il suffit d'indiquer l'adresse et le port du serveur. La création d'une socket côté serveur est un peu plus complexe car la socket doit être placée en écoute et attendre les connexions des socket client. Pour cela, on utilise la classe `ServerSocket` la socket serveur sera ensuite créée en utilisant la méthode `accept()`. Cette méthode est bloquante et attend la connexion d'un socket client.

Exercice : un logiciel client-serveur de discussion (chat) - version de test

Client v1

La première version du client est extrêmement simple, voici les opérations qu'il réalise :

- ouvrir une socket TCP/IP vers un hôte passé en paramètre (arg[0]) ;
- récupérez le flux de sortie caractère pour envoyer le message passé en second paramètre (arg[1]) ;

— lisez le message envoyé par le serveur.

On supposera qu'un serveur est en fonction. Pour les tests, le port et l'adresse vous seront communiqués en TP.

Serveur v1

Voici les opérations que devra réaliser votre serveur, après avoir créé une socket en mode serveur (`ServeurSocket`), les opérations suivantes seront réalisées dans une boucle infinie :

— Placer la socket en mode écoute (méthode accept `accept`) ;
— lire le flux envoyé par le client ;
— attendre pendant 5 secondes ;
— écrire "bonjour" dans le flux de sortie.

Serveur v2

La version précédente du serveur ne permet de traiter qu'un seul client à la fois (les autres sont placés en attente). Vous allez maintenant améliorer ce serveur pour qu'il puisse répondre simultanément à plusieurs requêtes de clients , en utilisant la programmation multi-threadée. Pour cela,

— écrivez la classe `Service` qui hérite de `Thread`, elle comportera un attribut privé de la classe `Socket` ;
— un constructeur pour initialiser cet attribut ;
— une méthode `run()` qui regroupe les opérations de lecture et écriture sur la socket (les mêmes que précédemment).

Ensuite, écrivez la classe `ServeurMultiThread` qui lancera un thread pour chaque client connecté.