Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Sockets
TP01

TAPR2 - Rapport

Filière informatique

Classe I3

Auteurs :

Jocelyn Duc
jocelyn.duc@edu.hefr.ch

Andrea Marcacci
andrea.marcacci@edu.hefr.ch

Enseignant :

Dr Rudolf Scheurer
rudolf.scheurer@hefr.ch

Fribourg, 30 octobre 2012

Table des matières

P0

Indiquer les systèmes d'exploitation utilisés et les versions Java installées des deux côtés (client et serveur)

Serveur

```
Mac OSX 10.7.5
java version 1.6.0\_35
Java(TM) SE Runtime Environment (build 1.6.0\_35-b10-428-11M3811)
Java HotSpot(TM) 64-Bit Server VM (build 20.10-b01-428, mixed mode)
```

Client

```
Mac OSX 10.7.5
java version 1.6.0\_35
Java(TM) SE Runtime Environment (build 1.6.0\_35-b10-428-11M3811)
Java HotSpot(TM) 64-Bit Server VM (build 20.10-b01-428, mixed mode)
```

P1

A quelle moment exactement (c.-à-d. à l'exécution de quelle méthode) la connexion TCP est-elle établit ? Répondre pour les deux côtés, client et serveur.

Pour répondre à cette question, il faut utiliser le debugger et éxécuter le client et le serveur en mode pas à pas afin de déterminer à partir de quelle méthode ces deux applications peuvent communiquer via les sockets.

Client

La connexion a lieu au moment de la création du socket. Il est possible de créer un socket de le préparer et de demander la connexion avec la méthode `connect()`

Serveur

Pour le client, la connexion au niveau Java est effective dès l'éxécution de la méthode `accept()`. Dès que le `ServerSocket` est instancié, le client peut s'y connecté en créant un socket, mais les données reçues sur le coté serveur ne peuvent être traîtées.
La méthode `accept()` est bloquante. C'est à dire que l'éxécution de notre boucle, côté serveur, va continuer dès qu'un socket client essayera de s'y connecter et, par ce biais, débloquer la méthode `accept()`.

P2

Est-ce que c'est nécessaire de spécifier le numéro de port TCP local côté client ? Si oui, comment peut-on le faire ? Si non, comment est-ce qu'on le ferait si on voulait quand-même le faire ?

Le client n'a pas besoin de spécifier son port local. Seuls l'adresse IP distante et le port distant sont nécessaire pour la connection.
Il est toutefois possible de forcer un port local. Pour ce faire, il faut utiliser un constructeur de la classe `Socket` qui permet ceci. Voici un exemple du code en question où le port local du client à été forcé à 61234 :

```
1  public TCPClient(InetSocketAddress isa) throws IOException {
2    InetAddress client = InetAddress.getByName("160.98.61.161");
3    InetSocketAddress isaClient = new InetSocketAddress(client,
         61234);
4    s = new Socket(isa.getHostName(), isa.getPort(), isaClient.
         getAddress(), isaClient.getPort());
5    oos = new ObjectOutputStream(s.getOutputStream());
6  }
```

Voici le résultat perçu du côté serveur :

```
Listening on TCP port 1234 ...
>> 0 : New connection request from /160.98.61.161:61234
@@ 0 : Test
## 0 : Connection closed
```

P3

Quelle est l'effet de la fermeture du ServerSocket (côté serveur), aussi par rapport aux connexions existantes ?

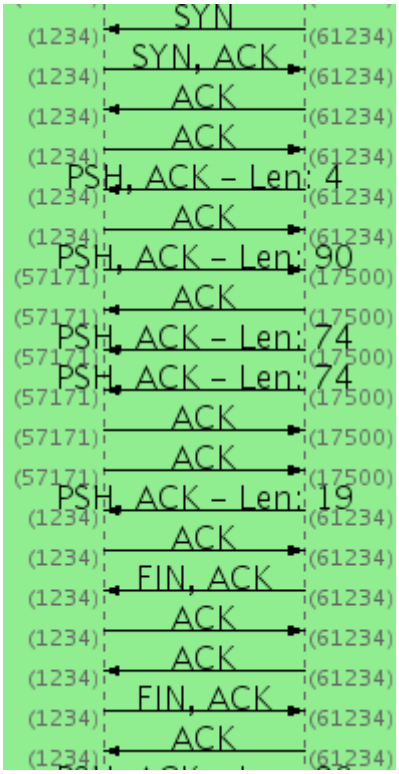
Une fois que le `ServerSocket` a dispatché les connexions vers les `TCPServerThread` (dans notre exemple), il peut être fermé sans couper les connexions entre les `Socket` des clients et les `TCPServerThread` sur le serveur.

Ceci n'a donc pas d'effet sur les connexions déjà établies. Par contre, il ne sera pas possible d'en établir des nouvelles.

P4

Faire une mesure contenant l'établissement, l'envoi d'un seul message et la fermeture de la connexion. Documenter (diagramme en flèche) et expliquer les trames entre l'établissement et la fermeture de la connexion.

Sur le diagramme suivant, nous constatons un échange de message via TCP. Le diagramme en flèche montre l'établissement via le 3-ways handshaking (`SYN, SYN-ACK, ACK`), l'envoi du message avec `PUSH-ACK` et `ACK` et la fermeture de connexion avec `FIN-ACK` et `ACK`



P5

Documenter et expliquer le format (de la charge utile TCP) utilisé pour l'envoi du message vers le serveur.

Les messages sont envoyés et reçus sous forme d'objet de type `String`. Les méthodes `readObject()` et `writeObject()` permettent de faire transiter des objets Java.

Le principe de la sérialisation Java permet d'enregistrer l'état d'un objet de façon persistance et de ce fait, de l'envoyer sur un flux à travers le socket du côté client et ainsi, de le restituer du côté du serveur.

P6

a

Que se passe-t-il si le client envoi des messages avant que le serveur exécute la méthode accept() ?

Sans exécution de la méthode `accept()`, le client peut tout de même créer un socket et y écrire des messages. Les messages seront immédiatement envoyés au serveur et y resteront en attente dans un buffer. Dès que l'on exécute la méthode `accept()` sur le

serveur, on récupère les sockets ouverts par le client quelques temps auparavant. Dans le cas de l'exercice, c'est une fois que les threads sont lancés que les messages sont consommés et affichés.

En conclusion, on peut dire que les messages ne sont pas perdus mais bel et bien stockés en attendant qu'on ne les consomme.

b

Repérer et expliquer les tailles des fenêtres TCP lors de l'envoi de plusieurs messages consécutifs (toujours avant que le serveur exécute la méthode accept()). Documenter avec un diagramme en flèche.

La taille de la fenêtre diminue au fur et à mesure que l'on reçoit des messages que l'on ne consomme pas avec la méthode `accept()`. C'est-à-dire que le buffer se rempli, donc la taille de fenêtre diminue. Cette taille est indiquée au client dans le champs window. La taille de fenêtre varie par 256 bytes.

c

Quel est l'utilité de la méthode setReceiveBufferSize() dans ce contexte ?

La méthode `setReceiveBufferSize()` sert à spécifier une taille de buffer diffeérente pour le socket concerné. On pourra donc, grâce à cette methode avoir une taille de fenêtre plus grande ou plus adaptée aux besoins du socket.

P7

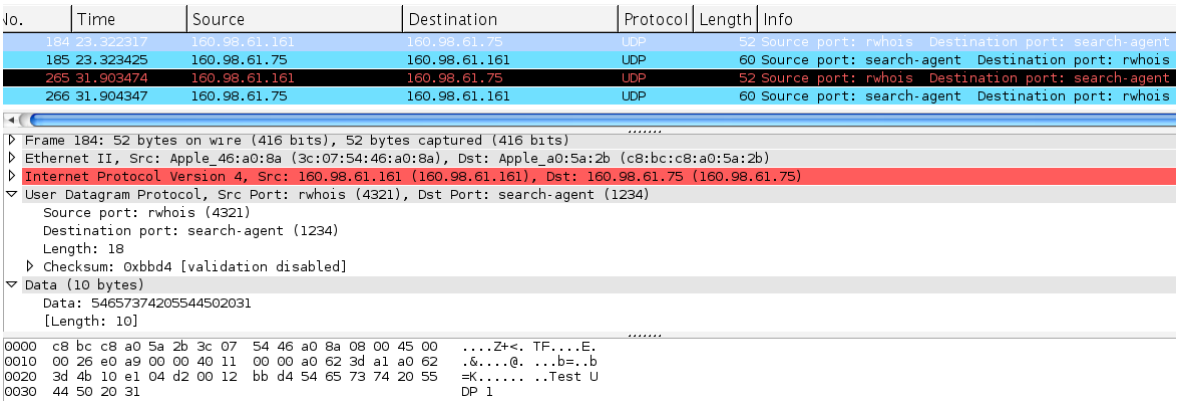
Sans utiliser les flux d'objets (Object...Stream etc.), comment pourrait-on transmettre des messages en format texte (pouvant contenir des retours à la ligne) ?

Il serait possible d'envoyer des messages en utilisant la classe `OutputStream` et également `StringBufferOutputStream` qui sont capables d'envoyer des `String` directement grâce à la méthode `write()`.

P8

Retrouver le format du message et vérifier le contenu et les ports UDP utilisés

Le contenu, les adresses et les ports se retrouvent clairement dans les trames capturées.



P9

Que se passe-t-il si le serveur n'est pas fonctionnel (port serveur pas ouvert) ?

Si le serveur n'écoute pas à ce moment-là, le serveur renvoie tout de même une erreur sous forme de message ICMP nous avertissant que le port est non-atteignable.

P10

Que se passe-t-il si l'émetteur émet son message avant que le récepteur appelle sa méthode receive() ?

Si le serveur reçoit un message avant d'appeler la méthode `receive()`, le message sera stocké. Il sera consommé dès la commande `receive()` appelée.