Programmation avancée Java

TP01

Professeur : R Scheurer

Auteur : Sébastien Bouquet

# Point 0

Les tests ont été effectués avec une machine cliente sous Fedora 26 et Java 1.8.0\_121 et une machine serveur sous Windows 10 et Java 1.8.0\_141

# Point 1

La connexion est établie lorsque le client crée un socket vers l’adresse du serveur. On peut voir, lorsqu’on utilise Wireshark, qu’après l’exécution de la création du nouveau Socket, le « three way handshake » a lieu entre le client et le serveur.

Le serveur ne connait par contre pas encore son client avant que celui-ci ait ouvert le ObjectOutputStream, ce qui a pour effet d’envoyé un premier packet de données (flags [PSH, ACK]) vers le serveur.

# Point2

Il n’est pas nécessaire de préciser le port TCP local du client, l’OS va automatiquement choisir un port TCP libre si on ne le précise pas.

On peut le préciser lors de la création du Socket du côté client :

*« new Socket(InetAddress, serverPort, localAddr, localPort) ;*

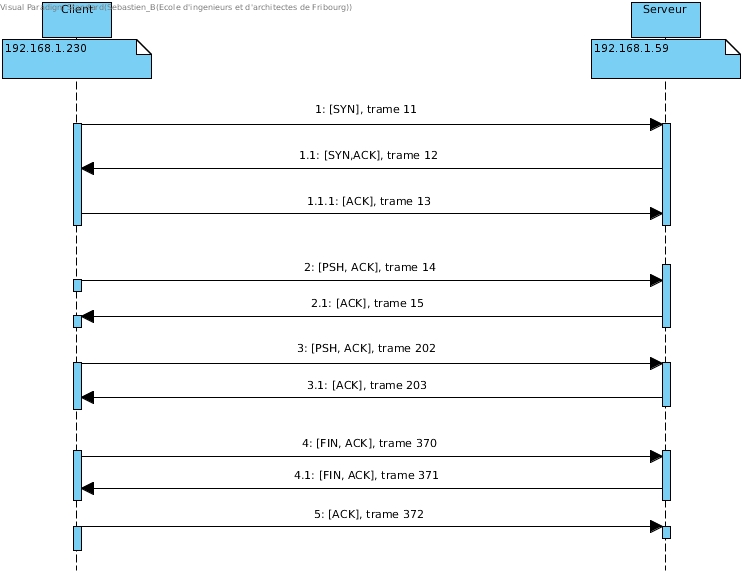
# Point 3

Cette action va lancer une *IOException*.

# Point 4

Le client ne s’aperçoit pas quand le serveur est éteint, bien que celui-ci envoie un message [RST, ACK]. Par contre si le client essaie d’envoyer un message, une *SocketException* sera levée.

# Point 5



Les trois premières trames correspondent à l’établissement de la connexion entre le client et le servuer.

Les deux trames suivantes correspondent à l’ouverture du *ObjectOutputStream.* Les deux suivantes sont un envoi de message du client au serveur.

Finalement les trois dernières trames correspondent à la fermeture de la connexion initiée par le client.

# Point 6

Dans la charge utile, on peut remarquer quel objet Java a été utilisé pour transmettre les données, en l’occurrence un *Java/lang/String*.

# Point 7

Les messages envoyés sont stockés dans le buffer du serveur et sont lu au moment où la méthode *receive* est lancée. Cela confirme notre réponse du point 1 puisque la connexion à bien due être établie avant cette méthode pour que les messages puissent être stockés dans le buffer.

# Point 9

Les caractères accentués sont représentés par plusieurs caractères ASCII. Par exemple un ‘è’ est représenté par les caractères ASCII 0xc3 et 0xa8 qui correspondent à « ├ » et « ¿ ».

# Point 10

Dans l’envoie de notre classe *MyMessage* on peut apercevoir le nom de la classe transmis dans les données TCP.

# Point 12

Le format du message est uniquement des caractères ASCII. Les ports sont affichés dans les trames Wireshark.

# Point 13

L’application client ne peut pas remarquer si le port est fermé ou si le serveur ne répond pas puisqu’il n’y a pas d’initiation de connexion pour le protocole UDP et que les messages envoyés ne sont pas acquittés par le serveur. Le client ne peut donc pas tirer la conclusion que ses données sont perdues parce que la machine distante ne répond rien.

# Point 14

Les messages envoyés sont stockés dans le buffer de réception. Ils seront lus lors du prochain appel à la méthode *receive* (en FIFO) à condition que le buffer ne soit pas plein.