Protocole TCP

Exercice 1 - Un client Echo en TCP

Écrire un client Echo s'appuyant sur TCP. Il faut pour cela établir une connexion avec un serveur donné, sur son port TCP 7, en créant un objet de la classe `Socket` qui établit la connexion. On peut alors lire ou écrire sur les flots d'entrée ou de sortie associés à cet objet. La connexion se termine en fermant la socket par la méthode `close()`.

Exercice 2 - Serveur de mise en majuscules

Écrire un programme, `UpperCaseTCPServer`, représentant un serveur TCP itératif revoyant les chaînes de caractères envoyées par des clients après les avoir mis en majuscule.
Ce serveur crée une socket serveur (objet de la classe `java.net.ServerSocket`), puis est démarré. Il attend alors une connexion (dite **pendante**) d'un client, via la méthode `accept()` appelée sur l'objet `ServerSocket`. Lorsque des clients contactent le serveur, l'une des connexions correspondantes est élue par la méthode `accept()`, qui retourne un objet de la classe `Socket`. Celle-ci est alors dite **socket de service**. Le serveur peut ainsi satisfaire la ou les requêtes successives émises par le client sur la socket de service. Puisque notre serveur est **itératif**, lorsque les différentes requêtes de ce client sont traitées, la socket de service peut être fermée, et une nouvelle connexion pendante peut être élue comme socket de service par un appel à `accept()` sur l'objet `ServerSocket`.
Le principe du serveur `UpperCaseTCPServer`, pour le traitement des requêtes d'une socket de service qu'il vient d'accepter, est le suivant:

1 Envoyer au client un message d'accueil, précisant les modalités d'envoi des requêtes. Par exemple, les chaînes doivent être envoyées ligne par ligne, et l'envoi d'une ligne contenant uniquement un point ("`.`") termine la session.
2 Pour chaque ligne reçue par le client, le serveur la met en majuscule et la renvoie au client.
3 Lorsqu'une ligne ne contenant qu'un point est reçue, le serveur renvoie cette ligne puis ferme la socket de service : il termine ainsi sa session avec ce client.

À son lancement, le serveur affiche son adresse et son numéro de port d'attachement local, afin que les clients puissent accéder à son service. Utiliser le client `netcat` pour effectuer les tests du serveurs.

Exercice 3 - Serveur UpperCase concurrent

Reprendre les spécifications du serveur `UpperCaseTCPServer` et écrire un serveur `UpperCaseTCPConcurentServer` qui délègue le traitement des requêtes correspondant aux sessions établies avec chaque client à des processus légers distincts.

1 Ècrire dans un premier temps, une version créant un `Thread` pour le traitement de chaque client.
2 Quel est le problème d'une telle architecture ?
3 Changer le code du serveur pour que les `Thread` soient créés avant l'acception d'une connexion d'un client. On pourra par exemple créer 10 processus légers. Lors de

l'acception d'une connection, un processus leger vacant recevra la socket de service qu'il devra traitée. Lorsque les 10 processus légers sont occupés, les clients ne peuvent plus être servis tant que l'une au moins des sessions établies n'est pas terminée.

Exercice 4 - Proxy X

Lorsque l'application `X` (xterm) d'une machine cliente `C` désire s'exporter sur une machine distante `D`, il arrive que cette connexion soit interdite par un firewall qui refuse de laisser passer les connexions de `C` vers le port `6000` (système de fenêtrage `X`) de `D`. Dans ce cas, le recours à un relai applicatif s'exécutant sur une machine `P`, dite **proxy**, peut être une solution. Dans ces conditions, une connexion depuis `C` vers le port `6001` de `P` peut donner lieu à l'ouverture d'une connexion entre le proxy et le port `6000` de la machine distante `D` (il faut bien entendu que le proxy ait le droit de se connecter sur ce port, ce qui est son rôle). Après cet établissement de connexion, le proxy crée et démarre deux processus légers chargés de relayer les informations circulant, d'une part, entre le client et la machine distante et, d'autre part, entre la machine distante et le client.
Écrire l'application `ProxyRelay` à installer sur la machine `P` comme suit:

```
machineP$ java ProxyRelay machineD 6000
Relai Proxy installé sur machineP:6001
```

La machine cliente pourra vérifier son fonctionnement en exportant un nouveau **display** et en ouvrant un xterm sur la machine distante par l'intermédiaire du proxy:

```
machineC$ export DISPLAY=machineP:1
machineC$ xterm
```