Programmation réseaux

TP 7
Serveur TCP élaboré avec threads en Java

Mars 2007

Le but de ce TP est de revoir les principaux concepts (threads et TCP en java) vus jusque là en programmant un nouveau serveur avec un protocole très simple.

Quelques rappels de Java sont placés en annexe.

Exercice 1 – *Serveur echo*

1. Programmer un serveur qui répète intégralement au client ce qu'il lui dit. Le serveur coupera la connexion quand le client lui enverra la chaîne `"\quit"`. Interrogez-le avec telnet.
2. Modifier ce serveur pour qu'il soit capable de dialoguer avec plusieurs clients simultanément. Pour cela, on utilisera un thread par client. Interrogez le serveur avec plusieurs telnets.
3. Implémentez un compteur du nombre de clients connectés. À chaque connexion et déconnexion, le serveur affichera sur sa sortie standard le nombre de clients. Attention l'accès au compteur doit être réalisé par l'intermédiaire de méthodes synchronisées, pour éviter les accès simultanés.

Exercice 2 – *Contrôle du serveur*

Nous allons maintenant créer un thread pour permettre de contrôler le serveur. Ce thread, lancé au démarrage du serveur, attend des commandes tapées au clavier. Pour l'instant nous implémenterons juste la commande `"\end"` qui arrêtera le serveur.

Exercice 3 – *Table des clients*

Nous voudrions maintenant pouvoir différencier les clients en tenant à jour une table des connexions en cours. Pour commencer, nous allons placer les clients dans un tableau, et on limitera à la taille du tableau le nombre de clients possibles pendant la durée de vie du serveur.

1. Modifiez votre programme pour insérer chaque connexion dans le tableau.
2. Implémentez la commande serveur `\kill`, permettant d'arrêter une des connexions en cours. Exemple : `\kill 3` arrête la troisième connexion. Pour arrêter la connexion, on appellera la méthode `shutdownInput()` de la socket, ce qui fera que le `read()` renverra **`null`** (fin de flux), et dans ce cas, on fermera la socket avec `close()`.

Exercice 4 – *Pour aller plus loin : messagerie instantanée*

1. Un tableau de connexions ne paraît pas très adapté puisque les connexions peuvent disparaître à n'importe quel moment, et on aimerait que la mémoire soit libérée. Remplacer le tableau de connexions par une table d'association. Voir en annexe comment utiliser les tables d'association en java.

2. Implémentez la commande client `\name` permettant à un client de changer son nom. La table d'association associe maintenant un nom à chaque connexion au lieu d'un numéro. Lorsqu'un client veut changer son nom, on vérifie d'abord que le nom n'est pas déjà utilisé, puis on remplace la connexion dans la table d'association.
3. Implémentez une méthode permettant d'afficher un message sur tous les clients.
4. Informez tous les clients dès qu'un utilisateur se connecte, se déconnecte, ou change de nom.
5. Implémentez la commande client `\to` permettant d'envoyer un message privé à un autre client. Exemple :

   ```
   \to toto
   Salut toto
   ```

   (On pourra évidemment supprimer l'echo du premier exercice).

A Quelques rappels de Java

A.1 Sockets

```
ServerSocket ss = new ServerSocket(22222);
Socket s = ss.accept();

BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintStream ps = new PrintStream(s.getOutputStream());
String s = br.readLine();
ps.print("Salut")
```

L'écriture et la lecture sur une Socket se fait exactement comme dans un fichier ou sur les entrées/sorties standard. Rappel :

```
BufferedReader entreeClavier = new BufferedReader(new InputStreamReader(System.in));
entreeClavier.readLine();
System.out.println("Ciao");
```

A.2 Chaînes de caractères

```
while (!requete.equals("\\quit")) {
  if (requete.startsWith("\\name ")) {
    String nom2 = requete.substring(6); // la fin de la chaîne
    ...
  }

String s = String.valueOf( entier );
```

Voir la documentation de la classe String pour plus d'information.

A.3 Tables d'association

```
Map table = Collections.synchronizedMap(new TreeMap());
// Ici on veut que les accès soient synchronisés
table.put("maclé", valeur);
v = table.get("maclé");
v = table.remove("maclé");
```

A.4 Iterateurs sur les tables

```
Iterator it = table.values().iterator();
// tables.values() est une collection contenant toutes les valeurs
// de la table.

// L'interface Collection sert à rassembler un ensemble d'objets,
// (List, Map, Set ...) et définit une méthode iterator()
// qui renvoie un itérateur sur cette collection. Par exemple :


while (it.hasNext()) {
  ((Connexion) it.next()).ps.println(s);
  // Ici les valeurs de la table ont un champ ps de type PrintStream
}
```