TP introduction à la programmation réseau

Le but de ce TP est de programmer une architecture client-serveur en utilisant les sockets.
Un programme serveur est exécuté sur une machine dite distante derrière un port en attente de connexion de programmes clients. Un programme client, qui est exécuté sur une autre machine, envoie une requête sur le programme serveur "distant". Le programme serveur distant traite la requête, retourne une réponse que le programme client affiche.

Préliminaire

Sur les machines du CNAM, vous pouvez créer 2 machines virtuelles. Pour cela, après vous êtes connecté sur le compte Unix, choisir le menu K puis
`Machines LXC | Machines LXC client-serveur-ad… | lancer-opensuse-admin`
Apparaît alors 2 consoles une jaune une rouge représentant 2 machines (virtuelles). Sur chacune de ces machines, connecter vous avec le couple d'authentification (`root`, `open`). Chaque machine virtuelle a sa propre adresse IP et ces deux machines peuvent communiquer entre elles par réseau.
A la création des 2 machines virtuelles, il est indiqué le nom DNS et le numéro IP des 2 machines virtuelles soit, par exemple :
`client … 10.0.3.7`
`server … 10.0.3.5`

1°) Tester que ces 2 machines virtuelles communiquent correctement par les commandes :
`ping nomDeLAutreMachineVirtuelle`
et
`ping numeroIPDeLAutreMachineVirtuelle`
On trouvera la valeur exacte de l'adresse IP d'une machine à l'aide de la commande :
`ifconfig`
On va écrire un programme client sur la "machine jaune" et un programme serveur sur la machine "rouge".

Il est proposé d'utiliser l'éditeur `Kate` accessible par menu `K | Editeurs | Kate`.
Après insertion d'un programme Java dans cet éditeur, vous pouvez avoir un bon environnement par :
a) indiquer que le texte inséré est du code Java par clic sur le bouton Normal (en bas à droite) puis Sources | Java. De la coloration syntaxique doit apparaître.
b) sélectionner tout le contenu par CTRL A, puis Outils | Indentation | Style C
c) Faire Outils | Aligner. Tout doit être bien indenté.

Anciennement on utilisait l'éditeur KWrite accessible par menu `K | Editeurs | KWrite`. Pour indenter entièrement un fichier, choisir `Outils | Indentation | Style C`, puis sélectionner son contenu en entier, et `Outils | Aligner`.
La documentation complète de KWrite se trouve à
`https://docs.kde.org/stable5/en/applications/katepart/index.html`

Ecriture du serveur

2°) Lancer un éditeur de texte et copier la trame du programme serveur ci-dessous :

fichier `MonServeurSimple.java`

```
-----------------------------------------
// Exemple inspire du livre _Java in a Nutshell_
import java.io.*;
import java.net.*;
public class MonServeurSimple {
	public final static int DEFAULT_PORT = 6789;
	protected int port;
	protected ServerSocket listen_socket;
	protected BufferedReader in;
	protected PrintStream out;
	// traitement des erreurs
	public static void fail(Exception e, String msg)
	{
		System.err.println(msg + ": " + e);
		System.exit(1);
	}
	// Cree un serveur TCP : c'est un objet de la
	// classe ServerSocket
	// Puis lance l'ecoute du serveur.
	public MonServeurSimple(int port) {
		if (port == 0) port = DEFAULT_PORT;
		this.port = port;
		Socket client = null;
		try {
			listen_socket = // A COMPLETER

			while(true) {
				client = // A COMPLETER
				in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
				out = new PrintStream(client.getOutputStream());
				traitement();
			}
		} catch (IOException e) {
			fail(e, "Pb lors de l'ecoute"); }
		finally { try {client.close();} catch (IOException
				e2) { } }
	}
	// Le lancement du programme :
	// - initialise le port d'ecoute
	// - lance la construction du serveurTCP
	public static void main(String[] args) {
		int port = 0;
		if (args.length == 1) {
			try { port = Integer.parseInt(args[0]); }
			catch (NumberFormatException e) { port = 0; }
		}
		new MonServeurSimple(port);
	}

	public void traitement() {
		String line;
		StringBuffer revline;
		int len;
		try {
			for(;;) {
				// lit la ligne
```

```
                        line = in.readLine();
                        // Si cette ligne est vide,
                        // le serveur se termine
                        if (line == null) break;
                        // sinon l'ecrit a l'envers
                        len = line.length();
                        revline = new StringBuffer(len);
                        for(int i = len-1; i >= 0; i--)
                              revline.insert(len-1-i, line.charAt(i));
                        // et l'envoie dans la socket
                        out.println(revline);
                  }
            } catch (IOException e) { ; }
      }
}
```

Un programme Java commence par la fonction (euh méthode statique) `main()` (si, si). Essentiellement cette méthode lance le constructeur

```
new MonServeurSimple(port);
```

Ce constructeur doit initialiser un objet de la classe `java.net.ServerSocket` qui écoute derrière le port `port`. Cet objet est alors référencé par la référence `listen_socket`.

3°) Ecrire l'initialisation de cette référence `listen_socket` (partie `// A COMPLETER`) en construisant un objet de la classe `java.net.ServerSocket` qui écoute derrière le port `port`. On pourra lire la documentation de cette classe à l'URL
`https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html`

4°) Par la suite, cet objet repéré par `listen_socket` se met en écoute derrière son port associé. Lorsqu'un client se connecte sur le serveur l'écoute est arrêtée, la socket qui mène au client est obtenue et le traitement de la requête est lancé.
Ecrire le lancement de cette écoute dans la seconde partie `// A COMPLETER`. Le code retour de cette écoute doit initialiser la socket repérée par la référence `client`. Il suffit de lancer la méthode appropriée qu'on trouvera à partir de
`https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html`

La suite du code serveur est donnée. Elle consiste à récupérer les parties de la socket qui permettent de lire et d'écrire dans cette socket et de lancer le traitement.

5°) Lisez et indiquez ce que fait ce traitement indiqué par la méthode `traitement()`.

6°) Sauvegarder ce fichier dans la machine `server` sous le répertoire `open`. Plus précisément, on choisira le menu `Fichier | Enregistrer sous…`
Puis le répertoire `Home > LXC_server_ROOT > home > open`
Le fichier doit obligatoirement avoir le nom `MonServeurSimple.java`

7°) Dans la machine `server` (fenêtre rouge) mettez-vous dans le répertoire `/home/open`, (euh il y a des chances pour que vous y soyez déjà cf. `pwd`), vérifier que vous avez bien ce ficher (`ls` ou `dir` ou …)

8°) Compiler ce fichier par
`javac -d . MonServeurSimple.java`

Il doit être créé un fichier `MonServeurSimple.class`

Ecriture du client

9°) Lancer KWrite et copier la trame du programme client ci-dessous :
fichier `MonClient.java`

```
--------------------------------
// Exemple inspire du livre _Java in a Nutshell_
import java.io.*;
import java.net.*;
public class MonClient {
	public static final int DEFAULT_PORT = 6789;
	public static void usage() {
		System.out.println("Usage: java MonClient <machineServeur> [<port>]");
		System.exit(0);
	}
	public static void main(String[] args) {
		int port = DEFAULT_PORT;
		Socket s = null;
		// initialise le port
		if ((args.length != 1) && (args.length != 2))
			usage();
		if (args.length == 1) port =
				DEFAULT_PORT;
		else {
			try { port = Integer.parseInt(args[1]); }
			catch (NumberFormatException e) {
				usage(); }
		}
		try {
			// Cree une socket pour communiquer
			// avec le service se trouvant sur la
			// machine args[0] au port port
			s = // A COMPLETER

			// Cree les streams pour lire et ecrire
			// du texte dans cette socket
			BufferedReader sin = new
					BufferedReader(new
						InputStreamReader(s.getInputStream()));
			PrintStream sout = new
					PrintStream(s.getOutputStream());
			// Cree le stream pour lire du texte a partir du clavier
			BufferedReader in = new
					BufferedReader(new
						InputStreamReader(System.in));
			// Informe l'utilisateur de la connection
			System.out.println("Connected to " +
					s.getInetAddress() + ":"+ s.getPort());
			String line;
			while(true) {
				// le prompt
				System.out.print("> ");
				System.out.flush();
				// lit une ligne du clavier
				line = in.readLine();
```

```
                        if (line == null) break;
                        // et l'envoie au serveur
                        sout.println(line);
                        // lit une ligne provenant de la socket,
                        // donc du serveur
                        line = sin.readLine();
                        // Verifie si la connection est fermee.
                        // Si oui on sort de la boucle
                        if (line == null) {
                              System.out.println("Connection ferme par le
serveur."); break; }

                        // Ecrit la ligne traite par le serveur et envoye
                        // par lui.
                        System.out.println(line);
                  }
            }
            catch (IOException e) { System.err.println(e);
            }
            // Refermer dans tous les cas la socket
            finally { try { if (s != null) s.close(); }
            catch (IOException e2) { ; } }
      }
}
```

10°) Comme indiqué par le code, ce programme soit être lancé par exemple par :
`java MonClient` *`nomOuAddIPMachineServeur`*
éventuellement avec un numéro de port (si, si lisez le début de ce programme).
Construire une socket vers la machine server indiquée sur le port `port` (partie `// A COMPLETER`) en récupérant dans ce programme la machine indiquée sur la ligne de commande par *`nomOuAddIPMachineServeur`*.
On pourra lire la documentation de la classe `Socket` à l'URL
`https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html`
Cette socket est alors repérée par la référence `s`.

11°) Lisez la suite du programme. Que fait ce client ?

12°) Sauvegarder ce fichier dans la machine `client` sous le répertoire `open`. Plus précisément, on choisira le menu `Fichier | Enregistrer sous…`
Puis le répertoire `Home > LXC_`**`client`**`_ROOT > home > open`
Le fichier doit obligatoirement avoir le nom `MonClient.java`

13°) Dans la machine `client` (fenêtre jaune) mettez-vous dans le répertoire `/home/open`, (euh il y a des chances pour que vous y soyez déjà cf. `pwd`), vérifier que vous avez bien ce ficher (`ls` ou `dir` ou …)

14°) Compiler ce fichier par
`javac -d . MonClient.java`
Il doit être créé un fichier `MonClient.class`

Exécution de l'architecture client-serveur

14°) Dans la machine `server` (fenêtre rouge) mettez-vous dans le répertoire `/home/open`, lancer le programme serveur par :

```
java MonServeurSimple
```

15°) Dans la machine `client` (fenêtre jaune) mettez-vous dans le répertoire `/home/open`, lancer le programme client par :

```
java MonClient server
```

Tout devrait bien fonctionner. Euh que doit-il se passer exactement ?

Bonus

16°) Ecrire des traces dans le programme serveur. Par exemple faire en sorte qu'il affiche la chaine reçue et la chaîne renvoyée au client.
17°) multithreader le serveur de sorte qu'il ait une architecture :

```
le serveur écoute
lorsqu'un client se connecte, une thread est créée pour traiter
sa requête
retour de l'écoute
```

Il faut utiliser un cours sur les threads en Java (voir à `http://cedric.cnam.fr/~farinone/CCAM/threads.pdf`) mais une solution complète se trouve à `http://cedric.cnam.fr/~farinone/IAGL/reseau.pdf`.