TP1 sur les sockets TCP

TP 1 : Un premier exemple : L'application Hello_World *Ecrire une application Client-Serveur "Hello-World" en mode TCP.*

TP 2 *Modifier le serveur Hello_ World pour qu'il accepte plusieurs clients en mode itératif (simple !)*

TP 3 *Ecrire un serveur parallèle (une classe Serveur.java) qu'on réutilisera dans la suite de ce TP qui consiste à éxécuter une application donnée en acceptant plusieurs clients en parallèle. Pour l'instant on ne se préoccupe pas de la classe Application. On suppose tout simplement l'exsitence de celle ci que l'on nommera Application.java.*

TP 4 *Ecrire un serveur Hello_ World Parallèle.*

TP 5 Réalisation d'un serveur http
On veut réaliser ici un serveur http qui écoute sur un port désigné et qui répond aux requêtes HTTP GET.

Principe du serveur http

Chaque serveur de la toile (serveur Web) a un processus serveur qui écoute sur le port 80 TCP dans l'attente de connexions clients qui sont en général des navigateurs (Mozilla, Netscape, ...). Quand une connexion est établie, le client envoie une demande et le serveur envoie une réponse que le client affiche. Le protocole qui définit les demandes et les réponses est le protocole HTTP. Les différentes étapes de fonctionnement sont :

a) L'utilisateur clique sur une morceau de texte appelée URL (Uniform Ressource Locator) de la forme hppp ://adresse_serveur : numéro_port/Chemin. où l'on dégage trois parties :

1. *le nom du protocole (http)*
2. *le nom de la machine sur laquelle se trouve la page ainsi que le numéro de port (généralement port 80)*
3. *Le nom du fichier contennant la page HTML.*

b) Le navigateur détermine l'URL à partir de la sélection.

c) Il demande au serveur DNS l'adresse IP du serveur

d) Le serveur DNS répond

e) Le navigateur établit une connexion TCP avec le serveur sur le port 80.

f) Il envoie la requête HTTP GET qui est de la forme GET chemin HTTP/1.1

g) Le serveur analyse la requête syntaxiquement et extrait à partir de celle ci le chemin du fichier html.

h) Le serveur envoie alors le fichier html indiqué par le chemin bloc par bloc.

i) La connexion TCP est libérée

j) Le navigateur affiche tout le texte

Pour écrire le serveur http on vous demande d'utiliser :

1. *La classe Serveur.java écrit précédemment.*
2. *de développer la classe Application.java qui réalise les fonctions du protocole http côté serveur (c'est à dire les étapes g,h et j).*
   Pour réaliser l'étape g on utiliser les lignes de code suivantes.

```
// On utilise dans ce qui suit l'API java.util.regex
qui permet d'utiliser les
 expressions régulières ou regex
// comme on les connaît déjà dans les langages de script
 de type shell.
//  Cette API permet de traiter des morceaux de texte
 particuliers ou motifs
comme un analyseur lexical.

// La methode Pattern.compile permet de compiler
 un motif et la méthode
 matcher permet de créer un comparateur. matchs
 permet de vérifier si
la requête donnée sous forme de texte correspond
à l'expression régulière, c'est à dire si elle est
 correcte syntaxiquement.
Dans le cas où la requête
est correcte alors on extrait de celle ci ce qui nous intéresse,
 c'est à dire le chemin du fichier html.
Matcher get = Pattern.compile("GET (/?\\S*).*").matcher(requête);
if(get.matches()){
requête = get.group(1);


// Les deux lignes de code qui suivent peuvent être
aussi utilisés pour
 afficher le fichier index.html dans le cas où le
 chemin indiqué est un répertoire.
if(requête.endsWith("/") || requête.equals(""))
requête = requête + "index.html";
```

ANNEXES

Le client HelloWorld

```
// Client.java
import java.io.*;
import java.net.*;
public class Client {
public static void main(String argv[]) throws Exception {

/*Le client se connecte au site donné en premier argument
et sur un numéro de port donné en deuxième argument*/

Socket emission = new Socket(argv[0], Integer.parseInt(argv[1]));

// Impression du numéro de socket
System.out.println("SOCKET = " + emission );

// Impression du numéro de port
System.out.println("Socket de connexion:" + emission.getLocalPort());

// Création du flux entrant
 BufferedReader in = new BufferedReader(
 new InputStreamReader(emission.getInputStream()) );


// Création du flux sortant
PrintWriter out = new PrintWriter( new BufferedWriter(
new OutputStreamWriter(emission.getOutputStream())), true);

// Envoi de la chaîne Hello_World
 String str = "Hello_World";
out.println(str);

// Lecture du message echo (Hello_World)
str = in.readLine();
System.out.println(str);

// Fermeture des flux sockets
out.close();
in.close();
emission.close();
}
}
```

Le serveur Helloworld

```
// Serveur.java
import java.io.*;
import java.net.*;

public class Serveur {
public static void main(String argv[]) throws Exception {

/*  Le serveur crée une socket d'écoute sur le port indiqué comme premier argument
ServerSocket ecoute = new ServerSocket(Integer.parseInt(argv[0]));

// Imprimer le numéro de port
System.out.println("\n Le serveur reçoit sur le port : "  + ecoute.getLocalPort());

// Le serveur est prêt à recevoir des requets du client
System.out.println(">>> Serveur prêt!! ");

// Le serveur crée une socket d'échange sock_com
Socket soc_com = ecoute.accept();

// Impression de l'adresse du numéro de port
System.out.println(" " + soc_com.getInetAddress());


// Le client qui s'est connecté est connu par getInetAddress et getHostName
InetAddress origin = soc_com.getInetAddress();
System.out.println("Connection venant de:
 " + origin.getHostName());


// Création du flux in qui permet de lire par ligne.
 BufferedReader in = new BufferedReader(
   new InputStreamReader(soc_com.getInputStream())
                              );

 // Création du flux out qui  possède toutes les opérations print classiques.

   PrintWriter out = new PrintWriter( new BufferedWriter(
  new OutputStreamWriter(soc_com.getOutputStream())),
                             true);


  String str = in.readLine(); // lecture du message

  System.out.println("Message reçu  = " + str);
  out.println(str);     // renvoi du message reçu écho
```

```
    in.close();
        out.close();
        soc_com.close();
   }
}
```

Le serveur parallèle

```
// serveur_parallele
import java.io.*;
import java.net.*;

public class serveur_parallele{

public static void main(String[] argv) throws IOException
{

ServerSocket ecoute = null;
try{
   ecoute = new ServerSocket(Integer.parseInt(argv[0]));
   }

catch(IOException e)
{System.err.println("Impossible d'ecouter sur le port indique");
         System.exit(1);
       }
// Acceptation de socket  client
Socket client = null;
while(true){
try{
    client = ecoute.accept();

 // Creation d'un fils par thread
 Thread fils = new Thread (new Application (client));

System.out.println(" Thread " + fils.getName() + "cree");

// Lancement du fils
       fils.start();
    }
catch(IOException e)
{
         System.err.println("Erreur de création de fils ");
               System.exit(1);
}

}

}

}
\end{verbartim}
```

La classe application

```
// APPLICATION

import java.io.*;
import java.net.*;
class Application implements Runnable {
    Socket sock_com;
    Application (Socket sock_com) throws SocketException {
this.sock_com = sock_com;
    }
    public void run() {
try {
 System.out.println("Adresse socket" + sock_com.getInetAddress());
System.out.println("Connexion venant de "+ sock_com.getInetAddress().getHostName());

 // Creation des flux
 BufferedReader in = new  BufferedReader(new InputStreamReader(sock_com.getInputStre
  PrintWriter  out = new  PrintWriter(new BufferedWriter (new OutputStreamWriter(soc

.... suite


}catch (IOException e) { System.out.println("Erreur E/S");}
```