TP 8 : Café des protocoles

Consignes générales :

- Un compte-rendu par binôme
- Justifiez vos réponses mais soyez concis.

Objectifs

Approcher les communications par le code

Pré-requis

Couches réseau et transport, ports de communication, Java

Introduction

Pour joindre une application s'exécutant sur une machine quelconque à travers le réseau, il est nécessaire de détenir les informations suivantes :

- l'adresse IP de l'hôte
- le protocole de transport utilisé par l'application
- le port de communication sur lequel l'application s'attend à recevoir des sollicitations

A partir de ces données, les applications vont créer des canaux leur permettant de « communiquer directement » les unes avec les autres. Ces canaux seront dénommés sockets et masqueront les détails des couches inférieures aux applications. En d'autres termes, à ce niveau de la pile protocolaire, les entités n'ont aucune connaissance du réseau sous-jacent (couches 1 et 2 totalement masquées).

Dans ce TP, nous utiliserons des sockets pour le protocole TCP. Le cycle de vie du socket sera donc décrit par la figure 1.

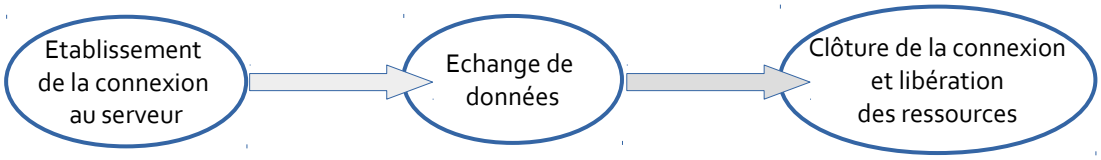

figure 4: Cycle de vie d'une socket (TCP)

Durant la phase d'échange des données, les applications « écriront/liront » dans le /du socket.

Activités

Exercice 1 Pour démarrer, transcrivez le programme P1 sur votre ordinateur et exécutez-le.

```
import java.net.Inet4Address;
import java.net.UnknownHostException;

public class P1 {
	public static void main(String[] args) {
		try{
			System.out.println(Inet4Address.getLocalHost().getHostAddress());
		}catch(UnknownHostException e){
			e.printStackTrace();
		}
	}
}
```

Vérifiez l'adresse retournée. Pourquoi l'accès à cette information se fait-il dans un try/catch ?

Exercice 2 A présent, modifiez le code de P1 pour obtenir le code P2 suivant :

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class P2 {
	public static void main(String[] args) {
		InetAddress addr;
		try{
			addr = InetAddress.getByName("192.168.1.42");
			if(addr.isReachable(1000)) System.out.println("Node 192.168.1.42 is reachable!!");
		}
```

```
                catch(UnknownHostException e){
                        e.printStackTrace();
                }
                catch(IOException i){
                        i.printStackTrace();
                }
        }
}
```

Remplacez l'adresse spécifiée en dur par l'adresse du poste utilisé.
En vous basant sur l'API, quel protocole est utilisé par la méthode isReachable(timeout_ms) ?

Exercice 3 Dans cet exercice, vous disposez de deux petits programmes, PA et PB. Chaque membre du binôme en choisira un qu'il retranscrira sur son ordinateur.
Quelle est la méthode invoquée par le serveur pour attendre une connexion entrante ? Combien d'instances du serveur peuvent tourner sur le même poste ?
Exécutez PA et PB sur deux postes différents. Quel est le numéro de port utilisé par le client ?

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;

public class PA {
        public static void main(String[] args) {
                ServerSocket server;
                Socket client;
                try{
                        server = new ServerSocket(4444);
                        client = server.accept();
                        System.out.println("aha!!");
                }
                catch(IOException i){
                        System.out.println("Impossible d'écouter sur le port 4444: serait-il occupé?");
                }
        }
}
```

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

public class PB {
        public static void main(String[] args) {
                InetAddress addr;
                Socket client;
                try{
                        client = new Socket("localhost", 4444);
                }catch(UnknownHostException e){
                        e.printStackTrace();
                }
                catch(IOException ioe){}
        }
}
```

TCP gère les octets générés par l'application comme un flux. L'API Java manipule ces flux à travers des objets de type *InputStream* et *OutputStream*, encapsulés dans un BufferedReader ou un PrintWriter. Ces flux sont rattachés à un socket, comme le montre le fragment de code suivant :

```
in = new BufferedReader( new InputStreamReader( client.getInputStream() ) );
out = new PrintWriter( client.getOutputStream(), true);
out.write( "pineapple\n");
```

Modifiez PA et PB pour échanger des messages entre le client et le serveur. Les messages seront saisis par les utilisateurs via le clavier. La fermeture de la connexion sera déclenchée par la réception de la chaîne de caractères «FIN».