IKHLEF Rafik
2303887
Groupe 4.

Compte Rendu TP1
Programmation de Sockets en java

Afin de communiquer entre plusieurs machines de manière simple, on utilise deux protocoles réseaux , le protocole TCP et le protocole UDP.

1-Le protocole TCP a été crée pour permettre d'envoyer des données de manière fiable par le réseau, notamment en :

-S'assurant que les données arrivent au destinataire, en cas d'échec de transmissio,, l'ordinateur émetteur doit être mis au courant.

-S'assurant que les données arrivent dans le bon ordre.

-Définissant une connexion entre les machines.

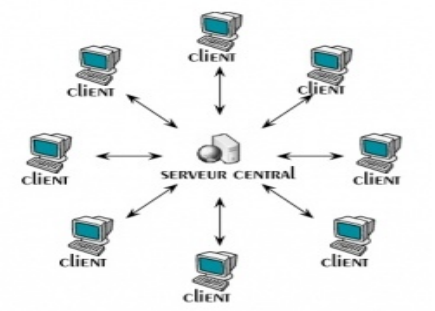
2-Le protocole UDP ajoute très peu au protocole IP. Par rapport au protocole TCP, UDP est peu fiable, il n'est pas certain que les données arrivent et il n'est pas certains que les données arrivent dans le bon ordre. TCP effectue un nombre important d'allers et retours, ce qui a l'inconvénient de faire diminuer la vitesse de connexion. De nos jours, TCP est quasiment tout le temps utilisé, sauf pour les échanges dont la perte de paquets n'est pas important (typiquement vidéocast, VoIP...).

- La classe Socket repose sur le protocole TCP et permet donc de transmettre des données de manière fiable. La classe DatagramSocket quant à elle, repose sur UDP.
- La plupart des systèmes d'exploitations proposent une abstraction pour manipuler des données réseaux que l'on nomme socket. un socket se comporte de la même manière qu'un fichier, Une fois une connexion établie, la gestion des sockets est similaire à la gestion des fichiers.

En java a été introduit une classe Socket qui permet de manipuler à plus haut-niveau cette abstraction du système d'exploitation. Il existe deux méthodes : getOutputStream() et getInputStream() qui permettent de manipuler les données comme pour un fichier.

-Java fait la différence entre socket côté client (objet qui permet de se connecter à des ordinateurs distants) et socket côté serveur (objet qui permet d'attendre des demandes de connexions de l'extérieur). La classe Socket correspond à un socket client et la classe ServerSocket correspond à un socket serveur.

-l'instanciation d'une socket permet de se connecter à un serveur nommé par son IP ou par son nom d'hôte sur un port particulier (new Socket(adress, port)). La notion de port, qui est purement virtuelle, a été définie au niveau de la couche TCP et UDP pour permettre de ne pas mélanger les données envoyées à une personne. Par exemple sur un chat, si vous souhaitez parler à quelqu'un et envoyer des fichiers à cette même personne en même temps, il est nécessaire de séparer les données.



Partie 1 :

exemple d'exécution :

```
bonjour client  : <127.0.0.1> : <57693>.
saisi un texte svp
coucou salut
COUCOU SALUT
saisi un texte svp
ca va toi
CA VA TOI
saisi un texte svp
oui bien merci et toi ?
OUI BIEN MERCI ET TOI ?
saisi un texte svp
tranquillo
TRANQUILLO
saisi un texte svp
ok
OK
saisi un texte svp
QUIT
QUIT. je dois fermer les portes. a bientot
```

Rep1

Il faut un objet de type « Socket » par service que la client pourra utiliser. Les applications client/serveur ne voient les couches de communication qu'à travers l'API socket. on a besoin d'une seule socket du coté serveur et de même pour le client.

Socket dite de service de la classe ***java.net.Socket*** qui établit la connexion TCP entre le client et le serveur de la classe Socket permet :

La connexion à un hôte distant, L'envoi/Réception de flot d'octets (notre message) et fermeture de la connexion.

ServerSocket de la classe **java.net.ServerSocket** qui est une socket TCP d'écoute (interface coté serveur en mode connecté) et se met en attente de connexion ,permet :

Un attachement à un port, Accepter les demandes de connexions au port local ,La gestion de l'attente de connexion des clients.

Rep2 : Le constructeur du socket client présenté en exemple en cours accepte deux arguments :

l'adresse distante en notation pointée , c'est-à-dire le *nom de la machine distante*, exemple : www.upmc.fr

le numéro de port distant visé (allant de 0 à 65535 mais jusqu'à 1023 réservés) : c'est un *identifiant local* pour connexion choisi en fonction de ce qui est « écouté » par le serveur.

exemple : 25 pour le protocole SMTP (envoi de mails), 23 pour le Telnet, 8080 pour le protocole HTTP .

-le port local peut être spécifié par l'application avec le constructeur. Pour montrer l'évolution de ce port, on peut utiliser la méthode int getLocalPort() de la classe Socket.

Rep3 : On n'utilise pas la méthode *readLine()* de la classe *DataInputStream* car elle ne convertit pas correctement les octets en caractères d'après la documentation. Elle est dépréciée. On doit donc lire les chaînes de caractères issues du socket par l'intermédiaire de la classe BufferedReader.

```
BufferedReader bistream = new BufferedReader(new InputStreamReader(s.getInputStream()));
```

Pour la lecture et l'écriture de données textuelles, java fournit les classes **BufferedWriter** et **BufferedReader** qui se construisent à partir de **OutputStreamReader** et de **InputStreamReader**. Cela offre les avantages des flux bufferisés et des méthodes supplémentaires pour gérer directement les chaînes de caractères.

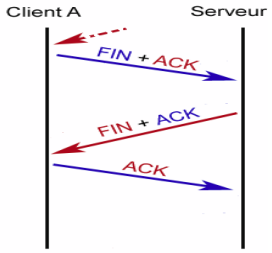
La classe **BufferedReader** dispose notamment d'une méthode **readLine()** qui permet de lire une ligne (donc finissant par \n). La classe **BufferedWriter** dispose d'une méthode **write** qui permet d'écrire directement une chaîne de caractères (au format String).

Rep4 -La primitive close() de la classe « socket » permet de fermer la « prise » proprement et suit la clôture de la connexion établie entre les deux entités. Cela libére la connexion donc le port utilisé. Il est nécessaire d'appeler cette primitive sur le client et sur le serveur pour éviter que le client (respectivement le serveur) tente de communiquer avec le serveur (respectivement le client) alors que le client (respectivement le serveur) a déjà fermé sa connexion.

-assurer la livraison des données avant la déconnexion (sinon exceptions).

-le **client** envoie au serveur un segment TCP avec le flag **FIN positionné à 1** (avec l'acquittement précédent si besoin)

-le **serveur** envoie a son tour un segment TCP avec le flag **FIN positionné a 1** avec l'acquittement précédent, **ACK positionné à 1,** le **client** acquitte avec **ACK à 1,** la connexion du coté serveur est libérée.



Lorsque l'on crée et que l'on connecte un socket, on a acquis des ressources données par le système d'exploitation, il est alors nécessaire de les rendre au système en utilisant la méthode, aussi bien sur les flux que sur les sockets. Celà est également utile pour la communication car on indique qu'on a terminé de parler.

Rep5 : Le cas où le **serveur ferme son socket alors que l'autre essaye de lire et/ou d'écrire sur le socket**, l'erreur générée est :
java.net.SocketException: Connection reset: Si le serveur a « brutalement » tué la connexion.
java.net.SocketException: Socket closed*:* Si le serveur ferme sa connexion via la primitive close() de la classe Socket.
La solution proposée à la question 4. est donc la méthode close() pour fermer. On devrait aussi faire le traitement de l'exception utiliser des catch(IOException e). Un try/catch autour de toutes les méthodes utilisant le socket de service (write . readline ...) est nécessaire.

Rep6 : Du point de vue TCP l'envoi vers le serveur **n'est pas réellement immédiat** lorsqu'un utilisateur veut envoyer un message (il appuie sur 'Entrée'). En effet, pour cela il faut l*e* flag PSH *(*push*)* doit être positionné à 1 pour indiquer au destinataire de remettre les données à l'application concernée dès leur arrivée Maximum Size Segment octets de données. les données seront envoyées au bout du *timeout défini par TCP.* La méthode **SetSoTimeout(int timeout)** permet de modfier le timeout.

Rep7 : La méthode SendUrgentData (public void sendUrgentData(int data) throws IOException) dans la classe « Socket » permet d'envoyer un octet de données urgentes sur le socket.
Le flag TCP manipulé par cette méthode est le flag ***URG***. Lorsque ce flag est à 1 les données doivent être traitées en priorité (urgente) pour l'émission.
A l'émission, il est envoyé *après les* données qui sont en train d'être transmises mais avant les données du tampon prêtes à être envoyées. **A la réception,** les données urgentes sont reçues de la même manière.

Rep8 : La méthode "setTcpNoDelay" dans la classe "Socket" *(*public void setTcpNoDelay (booleanon) throws SocketException) permet d'activer (on == true) ou non **l'algorithme de Nagle avec l'optionTCP_NODELAY.**
L'objectif de cet algorithme est d'améliorer l'efficacité du protocole en réduisant le nombre de paquets nécessaires à un transfert limitant les risques de congestion.
Le flag manipulé par cette méthode est **PSH (à 1 si désactivé).** Si il est activé, pour l'émission, il faut attendre l'acquittement du premier octet envoyé immédiatement au début (ou si la taille du tampon atteint la taille maximale d'un paquet) pour envoyer les données du tampon dans un seul paquet. Sinon on écrit simplement dans le tampon. Ainsi l'émission peut-être ralentie pour les petits paquets (petits face à Maximum Size Segment).

Voici le code de la partie 1 :

classe Client.java

```
 package TP1;
import java.net.*;
import java.io.*;
public class Client {
	public static void main(String[] args) throws IOException {
		BufferedReader bistream=null;
		BufferedWriter bostream=null;
		String ip,chaine,line = null;
		int port;
		boolean connecter=true;
		Socket s=null;
			if(args.length!=2){
				ip="localhost";port=12345;
			}else{
				ip=args[0];
				port = Integer.parseInt(args[1]);
			}
```

```
                try{
                        s= new Socket(ip,port);
                        bostream = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
                        bistream = new BufferedReader(new InputStreamReader(s.getInputStream()));
                        BufferedReader lireclavier=new BufferedReader(new InputStreamReader(System.in));
                        System.out.println(bistream.readLine());
                while(connecter){
                        System.out.println("saisi un texte svp");
                        chaine =lireclavier.readLine();
                   bostream.write(chaine +'\n');
                   bostream.flush() ;
                   if(!chaine.equals("QUIT")){
                   line = bistream.readLine();
                   System.out.println(line);
                   }else{
                        connecter=false;
                        line = bistream.readLine();
                        System.out.println(line+". je dois fermer les portes. a bientôt") ;
                   }
                }
                bostream.close();
                bistream.close();
                s.close();
                }catch(UnknownHostException e){
                        e.printStackTrace();
                }catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Classe Serveur.java

```
package TP1;
import java.net.*;
import java.io.*;
public class Serveur {
        public static void main(String[] args) throws IOException{
                ServerSocket serveur = null ;
                Socket serveursocket = null ;
                BufferedReader istream = null;
                DataOutputStream bostream=null,bostreamaccueil;
                String chainelue="";
                String chaineenvoie = "";
                int sport=12345;
                boolean connecter=true;
                try{
                serveur = new ServerSocket(sport) ;
                while(true){
                serveursocket = serveur.accept();
                bostreamaccueil=new DataOutputStream(serveursocket.getOutputStream());
                bostreamaccueil.writeBytes("bonjour client  : <"+ (serveursocket.getInetAddress()).getHostAddress()+"> : <"+serveursocket.getPort()+">.\n");
                connecter=true;
                while(connecter){
                istream = new BufferedReader(new InputStreamReader(serveursocket.getInputStream()));
                bostream = new DataOutputStream(serveursocket.getOutputStream());
                        chainelue = istream.readLine();
                        chaineenvoie=chainelue.toUpperCase();
                        if(!chainelue.equals("QUIT")){
                                bostream.writeBytes(chaineenvoie +'\n');
                                bostream.flush();
                        }else{
                        bostream.writeBytes(chaineenvoie +'\n');
                        bostream.flush();
                        connecter=false;
                        }
                }
                bostream.close();
                istream.close();
                serveursocket.close();
                serveur.close();
        }
                }catch(UnknownHostException e){
                        e.printStackTrace();
                }catch (IOException e) {
        e.printStackTrace();
                }
    }
}
```

Partie 2 :

voici le code des classes :

classe Servermulti.java :

```
package TP1.N_Client_serveur;
import java.net.*;
import java.io.*;
public class ServerMulti {
	public static void main(String[] args) {
		int idClient=0;
		int port=12345;
			System.out.println("server sur le prort : "+port);
			ServerSocket SocketServer = null;
			Socket socket=null;
			try{
				SocketServer=new ServerSocket(port);
				while(true){
				 socket=SocketServer.accept();
				//pour chque client, la prise en charge est ssurée par un thread
				ThreadServer thread=new ThreadServer(socket,idClient++);
				thread.start();
				}
			}
			catch(IOException e){
				System.err.println("Erreur : "+e);
			}
		}
}
```

Classe ThreadServer.java

```
package TP1.N_Client_serveur;
import java.net.*;
import java.io.*;
public class ThreadServer extends Thread {
	private DataOutputStream out;
	private BufferedReader in;
	private Socket socket = null;
	public int clientNo;
	public int reqcount = 0;
	String sentenceFromServer;
	  public ThreadServer ( Socket socket , int no ){
	   super("ThreadClient");
	   this.socket = socket;
	   this.clientNo = no ;
	  }
	  private String message_suivant() {
	     reqcount ++;
	     switch ( reqcount %5) {
	        case 0 : return new String ( " phrase 0 " ) ;
	        case 1 : return new String ( " phrase 1 " ) ;
	        case 2 : return new String ( " phrase 2 " ) ;
	        case 3 : return new String ( " phrase 3 " ) ;
	        case 4 : return new String ( " phrase 4 " ) ;
	     }
	   return new String ( " ca n'arrive jamais " ) ; //defaults
	  }
	  void pause () {
	   try {
	      Thread.currentThread() ;
	      Thread.sleep (1000) ;
	   }
	   catch (InterruptedException e ) {}
	  }
	  public void run() {
			try {
				InetAddress ip=socket.getInetAddress();
				int port=socket.getPort();
				out = new DataOutputStream(socket.getOutputStream());
				in =  new BufferedReader(new InputStreamReader(socket.getInputStream()));
			out.writeBytes("Bienvenue Client Numero "+clientNo+ " IP + "+ip+": Port  "+port+'\n');
				out.flush();
				while (!(in.readLine()).equals("QUIT")) {
			    out.writeBytes(message_suivant()+'\n');
			    out.flush();
				}
				out.writeBytes("je ferme la connexion pour le client "+clientNo+"\n");
				out.flush();
			  out.close();
```

```
                    in.close();
                    socket.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
```

---

classe ClientMulti.java

```
package TP1.N_Client_serveur;
import java.io.*;
import java.net.*;
public class ClientMulti {
        static void pause(){
                try{
                        Thread.currentThread();
                        Thread.sleep(2000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
        public static void main(String[] args) throws IOException {
                int port,compteur =0; // nombre d'echanges
                String ip;
                Socket socket = null;
                if(args.length == 2) {
                        ip = args[0];
                        port = Integer.parseInt(args[1]);
                }else {     ip = "localhost";
                                port = 12345;
                }
                try {
                        socket = new Socket(ip,port);
 BufferedWriter bostream = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
 BufferedReader bistream = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                        System.out.println(bistream.readLine());
                        while (compteur<10) {
                                compteur++;
                                String chaine="Je suis le client "+ip+" et j'ai fait "+compteur+" appels";
                                bostream.write(chaine+"\n");
                                bostream.flush();
                                pause(); //pause de 2sec
                                System.out.println(bistream.readLine());
                        }
                        bostream.write("QUIT\n");
                        bostream.flush();
                        pause(); //pause de 2sec
                        System.out.println(bistream.readLine());
                        System.out.println("QUIT");
                } catch (UnknownHostException e) {
                        e.printStackTrace();
                } finally{
                        socket.close();
                }
        }
}
```

---

exemple d 'exécution

```
             server sur le prort : 12345
 Bienvenue Client Numero 0 IP + /127.0.0.1: Port  59128
 phrase 1
 phrase 2
 phrase 3
 phrase 4
 phrase 0
 phrase 1
 phrase 2
 phrase 3
 phrase 4
 phrase 0
je ferme la connexion pour le client 0
QUIT
```