TP4 Programmation répartie
Sockets en Java

Safa YAHI

Exercice 1: Serveur UDP daytime

Considérons le serveur UDP daytime suivant :

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetSocketAddress;
import java.util.GregorianCalendar;

public class ServeurUdpdaytime {

	private int port;
	private int nbClients;

	public ServeurUdpdaytime(int port, int nbClients) {
		this.port = port;
		this.nbClients = nbClients;
	}

	public void lancer() {
		byte[] buf = new byte[512];

		try {

			DatagramSocket socket = new DatagramSocket(null);
			socket.bind(new InetSocketAddress(port));

			DatagramPacket packetReceive = new DatagramPacket(buf,
			buf.length);

			String chaineDate;
			System.out.println("Serveur daytime lancé sur : " + port);


			for (int i = 1; i <= nbClients; i++) {

				socket.receive(packetReceive);

				chaineDate = new
						GregorianCalendar().getTime().toString();

				DatagramPacket packetSend = new DatagramPacket(
						chaineDate.getBytes(),
						chaineDate.getBytes().length,
						packetReceive.getAddress(),
						packetReceive.getPort());

				socket.send(packetSend);
```

```
                }

                socket.close();

            } catch (IOException e) {
                e.printStackTrace();
            }

    }

    public static void main(String[] args) {

        new ServeurUdpdaytime(Integer.parseInt(args[0]),
                Integer.parseInt(args[1])).lancer();
    }

}
```

Questions :

1) Quelle est l'adresse de la socket du serveur ? Lancez-le et vérifiez votre réponse avec la commande “netstat -lun” (l pour listening, u pour udp et n pour numeric).

2) Considérons les deux instructions suivantes :

```
DatagramSocket socket = new DatagramSocket(null);
socket.bind(new InetSocketAddress(port));
```

Quel est le constructeur DatagramSocket() qui nous permet de faire la même chose sans appeler explicitement la méthode bind() ?

3) Rappelez la signification des paramètres ainsi que la valeur de retour de la fonction recvfrom() de l'API des sockets en C.

4) La fonction équivalente en java est la méthode receive() (de la classe DatagramSocket) qui prend comme paramètre un objet DatagramPacket. Qu'englobe un DatagramPacket à l'issu d'un appel receive () ? Comment peut-on accéder à ces différents éléments ?

5) Pour préparer un DatagramPacket pour un appel receive(), on utilise le constructeur de deux paramètres. Que représente ces deux paramètres ? A-t-on besoin d'utiliser le constructeur de quatre paramètres que nous avons vu en cours ?

6) Rappelez le profil de la fonction sendto() de l'API en C et comparez le à celui de la méthode send() en java.

7) Où-est ce qu'on spécifie l'adresse du destinataire pour envoyer un message udp en utilisant les sockets en java ?

8) Testez le serveur avec “nc -u adresse_serveur port_serveur”. Comme d'habitude, il faut envoyer quelque chose au serveur UDP pour avoir sa réponse.

9) Peut-on utiliser le même DatagramPacket pour l'envoi et pour la reception ? Comment ?

Exercice 2 : Client UDP daytime

1) Quelle est la différence entre un client UDP et un serveur UDP (parlons du cas général en dehors du Multicast)?

2) Ecrivez une classe **ClientUdpDaytime** qui implémente un client pour le service daytime en mode UDP et testez la.

3) Rappelons que pour recevoir un message en TCP, la taille du buffer intervient principalement pour optimiser l'échange avec le noyau. Sinon, pour lire un messag, on peut faire plusieurs read() et on a à chaque fois la condition d'arrêt : soit la déconnexion de l'autre comme en daytime et HTTP (sans les connexions persistantes), soit une caractère spécial à une certaine position pour désigner la dernière ligne à l'image des réponses du serveur en SMTP, soit une ligne qui se termine par "\n" comme pour Echo et l'exercice du chat alternatif du TP1, etc.
   En revanche, en UDP, la taille du buffer est plus critique pour recevoir un message. Si on utilise un buffer de taille plus petite que ce qu'il faut, le message sera tronqué et on ne pourra pas lire par la suite même avec un deuxième receive()...

   => Testez votre client avec un petit buffer (taille 10, par exemple).

   => Comment on choisit la taille du buffer UDP ?

Exercice 3 : Méthode setSoTimeout()

Testez le client précédent avec le serveur ayant pour adresse 139.124.187.29 et pour port 50013.

Vous avez constaté que votre client est resté bloqué (même si vous envoyez un message au serveur). En réalité, il n'existe aucun serveur UDP sur 139.124.187.29 qui écoute sur le port 50013 : le message envoyé par le client est "perdu". Par ailleurs, il n'y a pas d'accusés de recepetion en UDP. Donc, ce client passe à la méthode receive() qui sera finalement bloquante indéfiniment...

Un moment donné, il faut se débloquer...Pour celà, on peut utiliser la méthode **setSoTimeout()** de la classe **DatagramSocket.**

1) Reportez vous à l'aide de la classe DatagramSocket et lisez sa description:
   https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html#setSoTimeout-int-

2) Modifiez votre client en utilisant la méthode setSoTimeout() et testez le.
3) La méthode **setSoTimeout()** existe aussi dans les classes **Socket** et **ServerSocket** (mode TCP). Référez vous à la docmuntation officielle de Java et résumez son utilité pour les deux classes.

Exercice 4

Voyons le serveur suivant (**mais ne le lancez pas depuis votre machine**):

```
import java.io.*;
import java.net.*;
import java.util.*;
```

```
public class ServeurCitations {

	private Vector<String> tableauCitations;
	InetAddress adrGroupe;
	int portGroupe;
	int pause;

	public ServeurCitations(String fichier, String adrGroupe, int portGroupe,
int pause)
			throws IOException {
		tableauCitations = new Vector<String>();
		BufferedReader fichierCitations = new BufferedReader(
				new InputStreamReader(new FileInputStream(fichier)));
		String buf;

		while ((buf = fichierCitations.readLine()) != null)
			tableauCitations.add(buf);

		fichierCitations.close();

		this.adrGroupe = InetAddress.getByName(adrGroupe);
		this.portGroupe = portGroupe;
		this.pause = pause;
	}

	public void lancer() throws IOException, InterruptedException {

		DatagramSocket socketServeur = new DatagramSocket();

		DatagramPacket paquetSend;
		String msgSeveur;

		int i = 0;

		System.out.println("Serveur de citations lancé ");

		while (true) {

			msgSeveur = tableauCitations.get(i);

			paquetSend = new DatagramPacket(msgSeveur.getBytes(),
					msgSeveur.getBytes().length, adrGroupe,
					portGroupe);

			socketServeur.send(paquetSend);
			Thread.sleep(pause*1000);

			i++;
			if (i == tableauCitations.size())
				i = 0;

		}

		// socketServeur.close();
	}

	public static void main(String[] args) throws IOException,
			InterruptedException {
		ServeurCitations serveur = new ServeurCitations("citations.txt",
				"238.0.0.1", 2000, 4);
```

```
            serveur.lancer();
        }
}
```

1) Que fait-on dans le constructeur ?

2) Habituellement, un serveur doit avoir un port d'écoute fixe défini par l'utilisateur (voir serveur udp daytime) afin de permettre aux clients d'initier la communication avec lui. Quel est le port de la socket d'écoute utilisé par ce serveur ? Est-ce que celà pose problème et pourquoi ?

3) Ce serveur envoie périodiquement un message du Vector. A qui envoie-t-il ce message ? Par ailleurs, le serveur fait un seul send(), qui est-ce qui duplique ce message pour atteindre tous les destinataires ?

4) Implémentez le client correspondant et testez-le avec l'instance du serveur indiquée par votre ensaignant(e). Affichez les message reçus.

Exercice 5 : Messages UDP structuré

Implémentez une application client/serveur en UDP où le client peut poser trois types de requêtes au serveur par rapport à une adresse IP à savoir :

- determiner la classe (« A », « B », « C », « D » ou « E ») de l'adresse IP
- avoir l'adresse réseau associée à cette adresse IP
- avoir l'adresse de diffusion correspondante.

Le client pose une requête par execution.

Par exemple, le client peut envoyer au serveur l'adresse 139.124.187.19 et lui demander sa classe. Le serveur répond par classe B. Un autre exemple: le client peut lui envoyer l'adresse 8.8.8.8 et lui demander son adresse réseau, le serveur répond par 8.0.0.0.

NB: on ne gère pas ici l'adressage hors classe.

Vous pouvez utiliser la classe CAdresseIP disponible sur Ametice.

Exercice 6 : Chat en groupe en utilisant le Multicast

Pour le Multicast, n'utilisez pas une adresse de la plage 224.0.0.0/8.
Chaque étudiant doit utiliser l'adresse multicast spécifiée par son enseignant(e). Il peut bien entendu partager cette adresse avec ses voisins pour discuter avec eux.

Ecrivez un programme qui permet d'effectuer une discussion instantannée (chat) en groupe en utilisant la classe MulticastSocket. Ce programme doit donc permettre :

- de joindre un group Multicast (défini par une adresse IP et un port).
- de recevoir les messages envoyés par tout élément de ce groupe
- d'envoyer un message au groupe (l'emetteur d'un message le reçoit aussi car il fait partie du groupe).

Pour l'interface graphique, servez vous de la classe “FenetreChat.java” (disponible sur Ametice). Cette classe contient deux composants :

- un “JtextArea” pour afficher toute la discussion (messages reçus et messages envoyés)
- un “JtextFiled” pour saisir les messages à envoyer.

=> Testez ce programme en local
=> Testez le ensuite avec vos voisins pour discuter avec eux en groupe.

Exercice 7 : Chat entre clients deux à deux

Implémentez une application client/serveur en TCP qui permet à des clients de discuter entre eux (deux à deux et non pas en Multicast). Le serveur doit indiquer à chaque client l'ensemble des clients qui lui sont connectés (connectés au serveur). Tout envoi de messages entre clients doit passer par le serveur. En fait, quand un client A veut envoyer un message à un client B, le client A envoie d'abord son message au serveur qui se charge ensuite de l'envoyer lui même au client B.