**TP 1 : Sockets (UDP/TCP)**

• Question 1 : Détaillez, en 5 phases, le mécanisme Java de communication, coté serveur, entre

un client et un serveur utilisant des "stream sockets" (classe ServerSoket et Socket).

**Solution**

1- création d'un objet ServerSocket ServerSocket s = **new** ServerSocket(port, nbClientsMax)

2- attente indéfinie d'une connexion cliente Socket connection = s.accept()

3- obtention des flots d'E/S ObjectOutputStream input =**new** ObjectOutputStream(connection.getOutputStream()) ObjectInputStream input =**new** ObjectInputStream(connection.getInputStream())

4- phase de communication client/serveur via les objets ObjectInputStream et ObjectInputStream

5- fermeture de la socket s.close()

• Question 2 : Détaillez, en 4 phases, le mécanisme Java de communication, coté client, entre un client et un serveur utilisant des "stream sockets" (classe ServerSoket et Socket).

**Solution**

1- création d'un objet Socket Socket s = **new** Socket(adresseServeur,port)

2- obtention des flots d'E/SObjectOutputStream input =**new** ObjectOutputStream(s.getOutputStream()) ObjectInputStream input = **new** ObjectInputStream(s.getInputStream())

3- phase de communication client/serveur via les objets ObjectInputStream et ObjectInputStream

4- fermeture de la socket s.close()

• Question 3 : comment plusieurs requêtes simultanées peuvent-elles être gérées par le

serveur ?

**Solution**

Chaque requête est satisfaite par un thread déclenché par le serveur.

• Question 4 : Pourquoi les " Stream sockets " permettent-elles de récupérer les données envoyées dans le même ordre que celui de leur envoi ?

**Solution**

Les communications par " Stream sockets " sont fondés sur le protocole TCP (protocole fiable, avec ordre).

• Question 5 : Différences entre une transmission UDP et une transmission TCP.

**Solution**

UDP ne garantit ni l'ordre des messages ni leur délivrance

• Question 6 : Pourquoi une adresse IP multicast ne correspond-elle à aucune machine hôte ?

**Solution**

Un message vers une adresse IP multicast sera reçu par tous les membres du même groupe

multicast alors que chacun est hébergé par sa propre machine hôte.

• Question 7 : D'après vous, quel type de socket l'application " telnet " utilise-t-elle ?

**Solution**

Telnet est un protocole de type client-serveur basé sur TCP. Les clients se connectent généralement sur le port 23 du serveur. Il fournit un système orienté communication, bi-directionnel

• Question 8 : Quelles informations sont-elles nécessaires à un client pour créer une

" DatagramPacket " à destination d'un serveur ?

**Solution**

Le tableau de bytes comportant les données, la longueur du tableau, l'adresse IP du destinataire, le port sur lequel le serveur attend les données transmises

• Question 9 : imaginez une application qui permet un échange de messages entre seulement 2

personnes via l'Internet. Quel type de logiciel mettre en place sur chacun d'entre-eux ? Quel

protocole doit-on choisir ? Ecrire les algorithmes nécessaires

**Solution**

Chacun des 2 participants est à la fois client et serveur.On utilisera le protocole UDP car l'envoi de message ne nécessite pas la connexion du partenaire.

**Algorithme serveur** :

création de la socket datagramme en réception sur un port donné;

création d'un tableau de bytes pour la réception des paquets;

boucler sur{

attendre la réception d'un paquet;

décodage du paquet;

affichage du paquet;

}

**Algorithme coté client** :

création de la datagramme socket;

boucler sur{

convertir le message en tableau de bytes;

former un datagramme vers un serveur et un port;

envoyer le paquet;

}

**Question 10** :Supposons que vous avez notre client Upcase sur un hôte et le serveur sur un autre hôte.

a) Supposons que vous exécutiez le client TCP sans démarrer le serveur. Qu'est-ce qui se passe exactement et pourquoi?

b) Supposons que vous exécutiez un client UDP sans démarrer un UDP serveur. Ce qui est différent du cas de TCP et de Pourquoi?

c) Que se passe-t-il avec TCP et UDP si vous démarrez le serveur? après le client mais avant de taper une entrée?

d) Que se passe-t-il si vous connectez le client TCP à un site Web?serveur fonctionnant sur le port 80 au lieu du serveur Upcase

**Solution**

a) Si vous exécutez d'abord un client TCP, le client tentera d'établir une connexion TCP avec un processus serveur inexistant. La création du socket client échouera.

b) Avec UDP, le client n'établit pas de connexion avec le serveur. La création d'un socket fonctionne, mais l'envoi d'un datagramme échouera.

c) Avec TCP, créer à nouveau la socket échouera. Comme aucune connexion n’est établie avec UDP, tout devrait fonctionner correctement si vous exécutez d’abord le client, puis le serveur,

puis tapez une entrée sur le clavier.

d) Le client établira une connexion. Si l'utilisateur tape quelque chose qui n'est pas HTTP, le serveur répondra par un message d'erreur. Si l'utilisateur tape GET / HTTP / 1.0

le serveur fournit la page Web correspondante.

**Question 11** :

Les demandes de connexion TCP (SYN) sont buffered sur le socket du serveur. Le serveur procède alors à la négociation en trois étapes. Accept () renvoie la demande de connexion suivante où la négociation à trois est terminée. Les demandes avec et sans négociation à trois terminées aboutissent dans le même buffer.

1. Qu'advient-il d'une nouvelle demande si le buffer est plein?

b) Que se passe-t-il si un client envoie un grand nombre de faux SYN à un serveur sans jamais terminer les trois voies poignées de main (SYN flooding)?

**Solution**

a) La demande ne peut pas être servie et le paquet SYN est abandonné.

b) La mémoire tampon sur le serveur est pleine de paquets SYN où la négociation à trois voies ne se terminera jamais. Les autres clients ne peuvent pas accéder au serveur car la mémoire tampon pour les demandes de connexion est saturée.

Une meilleure implémentation consisterait à créer des tampons distincts pour les demandes de connexion non terminées et les demandes de connexion avec établissement de liaison à trois.

**Question 12 :**

a) Dans l'en-tête de transport d'un paquet, vous avez le port source et le port de destination. Le récepteur a besoin du port de destination pour le démultiplexage mais quoi le port source est-il utilisé?

b) Supposons qu'un serveur a un socket de serveur ouvert sur le port X pour les connexions TCP entrantes. Que se passe-t-il si sur le même serveur vous ouvrez un socket UDP pour recevoir datagrammes utilisant également le nombre X? Y a-t-il des conflits?

c) Est-il possible d'avoir plusieurs threads pour gérer les clients simultanément avec un serveur UDP? Si oui comment? Si non, pourquoi pas?

d) Qu'en est-il d'utiliser plusieurs threads et plusieurs Sockets pour recevoir comme avec TCP?

**Solution**

a) Le récepteur TCP a besoin du port source pour savoir où envoyer le SYN / ACK. Pour UDP, le port source n’est pas aussi important que pour TCP, mais il est généralement utilisé pour

déterminer où envoyer les réponses.

b) Il n'y a pas de conflits. Le système d'exploitation dispose de 2 ^ 16 ports pour TCP et de 2 ^ 16 ports distincts pour UDP. N'oubliez pas qu'une connexion est identifiée de manière unique par l'IP source et le port,

IP de destination et port, ET type de protocole.

c) Ce n'est pas un problème. Créez un nouveau fil et remettez le datagramme du client. Le thread peut ensuite traiter la demande en arrière-plan, puis envoyer la réponse.

d) C'est possible, mais il n'y a pas de socket serveur pour UDP. Ainsi, cela doit être fait "à la main". Créez un nouveau socket lors du premier contact client et renvoyez le nouveau numéro de port au client afin qu'il puisse y envoyer les datagrammes suivants. C'est rarement fait!

**Question 13 :**

Réécrivez et implémentez le serveur Java UDP en utilisant plusieurs threads.(Annotez simplement dans le code source avec quoi des modifications doivent être apportées.)

* Que mettez-vous dans un fil séparé? Pourquoi?
* Devez-vous utiliser un nouveau socket pour chaque client?

**Solution**

Il y a plusieurs manières correctes de faire ceci. Voici un exemple:

**UDP Server Multi Threads**

import java.io.\*;

import java.net.\*;

class UDPServerMultiThreads {

public static void main(String args[]) throws Exception {

DatagramSocket serverSocket = new

DatagramSocket(9876);

System.out.println("I am waiting on port 9876");

while(true) {

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new

DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

ServerThread serverT = new ServerThread(

receivePacket);

serverT.startSocket();}}}

**Server Thread**

import java.io.\*;

import java.net.\*;

public class ServerThread extends Thread {

private DatagramPacket receivePacket;

public ServerThread(DatagramPacket receivePacket)

throws SocketException {

this.receivePacket = receivePacket;}

public void startSocket() throws Exception {

byte[] sendData = new byte[1024];

DatagramSocket responseSocket = new DatagramSocket();

String clientSentence = new String

(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

System.out.println(port);

String capitalizedSentence = clientSentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket = new DatagramPacket(

sendData, sendData.length, IPAddress, port);

responseSocket.send(sendPacket);

responseSocket.close();}}

**Question 14** :

Considérons une application de transfert de fichier qui transmet des données à une vitesse constante (par exemple, l'expéditeur génère une unité de données à N bits toutes les k unités de temps, où k est petit.et fixe). De plus, quand une telle application démarre, elle restera allumée pendant une période relativement longue.

Quel protocole de transport est le plus approprié pour cette application, TCP ou UDP? Pourquoi?

Supposons que vous avez des transferts Web où chaque page correspond en un seul paquet. Quels sont les avantages et les inconvénients d'utiliser UDP au lieu de TCP?

**Solution**

TCP est préférable pour ce type de transfert de fichiers, mais ce qu'il faut utiliser n'est pas toujours aussi clair (en théorie, on pourrait utiliser HTTP avec UDP).

* TCP:
* Utile si l'application implique de longues sessions avec des exigences prévisibles de bande passante lisse
* Les données doivent être transmises dans l'ordre et sans pertes; L'architecture orientée connexion aide à éviter tout gaspillage significatif au niveau de l'application

* Les frais généraux et les retards de négociation à trois voies peuvent être amortis sur la durée des sessions
* UDP:
* Utile si l'application implique des transactions courtes avec seulement quelques paquets à envoyer (pas besoin de contrôle de flux, évitez la surcharge de la négociation à trois voies)
* La reprise sur erreur (par exemple, retransmission / commande) doit être effectuée au niveau application si nécessaire

**Allez plus loin : mlutiserver avec muticlient (correction avel le dossier ci-joint)**