2ème année DUT Informatique
IUT Aix-en-Provence

TP3 Programmation Répartie
Sockets TCP en Java

Safa YAHI

Exercice 1 (Client TCP Echo)

Dans cet exercice, nous nous intéressons au service “echo”. Comme “daytime”, il s'agit d'un service assez simple utilisé souvent pour illustrer l'implémentation de sockets. Par défaut, le port associé est le port 7.

Ecrivez une classe **ClientTcpEcho** qui implémente un client TCP en java pour le service **echo**
Plus précisément :

- Au niveau du client, l'utilisateur saisit une chaine de caractères (lecture depuis le clavier).
- Le client envoie ensuite cette chaine au serveur.
- Le serveur envoie une nouvelle chaine au client. En général, c'est la même chaine, mais ici on veut qu'elle soit transformée en majuscule.
- le client reccupère la nouvelle chaîne envoyée par le serveur et il l'affiche.

Ce processus est réitéré jusqu'à ce que l'utilisateur entre la chaine “quit” ou que l'un des deux se déconnecte.

Cette classe admet **deux attributs** : une chaine de caractères (hostname) qui représente l'adresse IP (sous format textuel) ou le nom de machine du serveur et un nombre entier (port) qui désigne le port sur lequel écoute le serveur.

Pour initialiser ces attributs, on utilise **un constructeur** de deux paramètres.
Cette classe possède une méthode **public void lancerBW()** où le client utilise la classe BufferedWriter pour envoyer les données.

=> Testez la classe ClientTcpEcho avec le serveur echo indiqué par votre enseignant(e).

Exercice 2 (Serveur TCP Echo)

Écrivez une classe **ServeurTcpEcho** qui implémente un serveur TCP echo. La chaîne retournée par ce dernier correspond à la chaîne envoyée par le client transformée en majuscule. On suppose que ce serveur traite un nombre donné de clients (soit nbClients) et ce séquentiellement (pas de threads pour le moment). Pour chaque client qui se connecte, affichez son adresse IP.

=> Testez cette classe avec un client Telnet ensuite avec le client TCP Echo que vous avez implémenté précédemment, en local et avec votre voisin(e).

Exercice 3 (Serveur TCP Echo Multi-threads)
Testez le serveur précédent avec plusieurs clients lancés en même temps. Que remarquez-vous ?

Le but de cet exercice est d'implémenter une nouvelle classe **ServeurTcpEchoMulti** qui représente un serveur TCP Echo permettant de traiter plusieurs clients en même temps en utilisant les threads : à chaque fois qu'un client se connecte, le serveur lance un thread pour le traiter (échanger des messages avec lui).

D'abord, créez une nouvelle classe qui s'appelle **TaskServerEcho.** Cette classe implémente l'interface Runnable et définit le traitement d'un client dans la méthode run(). Dans cette dernière méthode, on utilise la socket du client qui résulte du accept(), donc pensez à définir un constructeur qui prend cette socket comme paramètre.

Implémentez à présent la classe **ServeurTcpEchoMulti** en lançant, à chaque connexion d'un client, un nouveau thread.

Exercice 4 (Serveur TCP Echo avec pool de threads).

Rappelez les avantages des pools de threads vus en cours.

=> Créez une nouvelle classe **ServeurTcpEchoPool** qui se base sur l'utilisation de pools de threads pour traiter plusieurs clients en même temps.

=> Testez-la avec un pool de 3 threads, lancez jusqu'à 4 clients et observez ce qui se passe.

Exercice 5 (Client SMTP pour envoyer des Emails)

AVERTISSEMENT !!!

L'objectif de cet exercice est purement pédagogique. L'envoi de messages anonymes contredit la charte de l'Université.
N'ESSAYEZ SURTOUT PAS D'USURPER L'IDENTITÉ DE QUELQU'UN POUR ENVOYER UN MESSAGE, SINON VOUS VOUS EXPOSEREZ À DE LOURDES SANCTIONS ET DES POURSUITES JUDICIAIRES.
UN MAIL ANONYME PEUT ETRE TRAÇABLE.

Introduction à SMTP

Le but de cet exercice est d'écrire un client SMTP en Java en utilisant les sockets TCP.

Le protocole SMTP (Simple Mail Transfer Protocol) permet le transfert des mails depuis la machine de l'émetteur vers la machine qui héberge la boite email du destinataire. Ce protocole est décrit par la RFC 2821. Il est basé sur un échange bien défini de messages entre le client et le serveur.

Les messages (commandes) du client les plus fréquemment utilisés sont :

- EHLO : pour l'adresse IP ou le nom de la machine sur laquelle s'exécute le client
- MAIL FROM : pour le mail de l'émetteur
- RCPT TO : pour le mail du destinataire
- DATA: pour le sujet, le corp du message, etc. Notons que la partie DATA doit se termnier par un "." afin d'indiquer la fin du messsage.

Côté serveur, les réponses commencent par un code de trois chiffres. Les codes qui débutent par 4 ou 5 correspondent à des messages d'erreurs. Sinon, il s'agit de messages positifs.

Aussi, les réponses du serveur peuvent tenir sur plusieurs lignes. Afin de distinguer la dernière ligne des autres lignes, on utilise un caractère spécial après le code de trois chiffres. Plus précisément, pour toutes les lignes hormis la dernière, on met le caractère "-" (moins) après le code. Pour la dernière ligne, il s'agit du caractère "SP" (un blanc). Voici un exemple d'une réponse multiligne :

```
250-mx0.univ-amu.fr   //  première ligne (caractère "-" après le code)
250-8BITMIME          // deuxième ligne (caractère "-" après le code)
250 SIZE 20971520     // dernière ligne  (pas de caractère "-" après le code)
```

Prenons l'exemple suivant où on se connecte au serveur SMTP 192.168.1.90 via le port 25 pour envoyer un mail de **garfield** à **jon**. Les messages envoyés par le client au serveur sont en bleu. Les messages en vert sont ceux envoyés par le serveur.

```
Telnet 192.168.1.90 25

Trying 192.168.1.90...
Connected to 192.168.1.90.
Escape character is '^]'.
220 jupiter ESMTP Postfix (Debian/GNU)
EHLO jupiter
250-jupiter
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
MAIL FROM: garfield
250 2.1.0 Ok
RCPT TO: jon
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
FROM: garfield
TO: jon
SUBJECT: Lasagnes

Coucou Jon,
Je veux des lasagnes :)
Bye
.
250 2.0.0 Ok: queued as 24D3E462E
QUIT
221 2.0.0 Bye
Connection closed by foreign host.
```

Partie 1 : Se familiariser avec SMTP

Afin de tester cet exercice et aussi le suivant, un compte de messagerie vous a été créé sur la machine 139.124.187.23 (login et mdp précisés par votre enseignant(e)). Cette machine dispose d'un serveur SMTP (port 25) pour l'envoie de emails et aussi d'un serveur POP3 (port 110) qui permet de récupérer des emails.

Evolution, à l'image de Thunderbird ou de Icedove, est un client de messagerie que vous allez utiliser ici pour vérifier l'accès à votre compte et aussi l'arrivée des mails échangés.

=> Configurez votre client de messagerie Evolution en indiquant notamment votre login, votre serveur SMTP et votre serveur POP3. Un exemple pour garfield est disponible sur Ametice sous forme de captures d'écran. Pensez à cocher la case «conserveur les emails sur le serveur » pour pop3.

=> Envoyez un premier mail vers vous ou vers votre voisin(e) depuis Evolution, vérifiez que le destinataire l'a bien reçu.

=> Envoyez maintenant un autre mail via Telnet et en utilisant les commandes SMTP (comme l'a fait garfield plus haut). Vérifiez qu'il est bien reçu avec Evolution.

Partie 2: Implémentation de votre client SMTP

Implémentez la classe ClientSmtp qui admet le **constructeur** et la méthode **sendMail** suivants :

- **public ClientSmtp(String serveurSmtp, int port, String hostname)** où :
  - “serveurSmtp” est l'adresse IP ou le nom du serveur SMTP du domaine du destinataire
  - “port” désigne le port sur lequel écoute le serveur SMTP (par défaut, c'est le port 25).
  - “hostname” est l'adresse IP ou le nom de la machine du client SMTP.

- **public boolean sendMail(String from, String to, String subject, String body)** où :
  - “from” indique l'adresse email de l'émetteur du message
  - “to” indique l'adresse du destinataire
  - “subject” désigne le sujet du message
  - “body” indique le corps du message

Dans la méthode sendMail(), le dialogue entre le client et le serveur se fait à tour de rôle. Autrement dit, le client n'envoie pas toutes les commandes d'un seul coup. Il doit à chaque fois prendre en compte la réponse du serveur. S'il s'agit d'un message d'erreur, alors le client interrompe la connexion.

Testez votre classe avec votre compte ou celui de votre voisin(e) toujours sur la machine 139.124.187.23.

Exercice 6 : Client POP3 pour récupérer des Emails

Le protocole POP3 (Post Office Protocol) permet de récupérer des emails depuis un serveur de messagerie. Il se base sur le protocole TCP et utilise, par défaut, le port 110.

Il se définit par un dialogue entre le client et le serveur. Les réponses du serveur qui commencent par "-ERR" signalent des erreurs alors que celle qui commencent par "+OK" veulent dire que tout

va bien.

Récupérer un mail (défini par un numéro) depuis le serveur s'effectue comme suit :

1. Le client se connecte au serveur.
2. Le serveur lui envoie un message sur UNE SEULE ligne (soit pour signaler une erreur, soit pour dire que tout est ok).
3. Le client s'identifie en envoyant d'abord le message "**USER *nom_utilisateur***"
4. Le serveur répond par UNE SEULE ligne.
5. Le client envoie le message "**PASS *mot_de_passe***"
6. Le serveur répond encore une fois par UNE SEULE ligne
7. le client envoie le message "**RETR *numéro_mail***" qui permet de réccuprer le mail ayant comme numéro "*numéro_mail*".
8. Le serveur répond à ça par une première ligne qui commence par +OK s'il n y a pas d'erreur et envoie ensuite le mail demandé sur plusieurs lignes. La fin du mail est marquée par "."
9. Enfin, le client se déconnecte en envoyant le message "**QUIT**".

Voici un exemple de POP3 avec telnet où jon reccupère le mail envoyé par garfield.

```
safa@jupiter:~$ telnet 192.168.1.90 110
Trying 192.168.1.90...
Connected to 192.168.1.90.
Escape character is '^]'.
+OK Dovecot ready.
USER jon
+OK
PASS progrepa
+OK Logged in.
LIST
+OK 2 messages:
1 424
2 496
.
RETR 1
+OK 424 octets
Return-Path: <garfield@jupiter>
X-Original-To: jon
Delivered-To: jon@jupiter
Received: from jupiter (etoile [192.168.1.90])
   by jupiter (Postfix) with ESMTP id 24D3E462E
   for <jon>; Mon, 13 Feb 2017 18:02:30 +0100 (CET)
FROM: garfield@jupiter
TO: jon@jupiter
```

Attention : mdp en clair sur Telnet:(

```
SUBJECT: Lasagnes+
Message-Id: <20170213170237.24D3E462E@jupiter>
Date: Mon, 13 Feb 2017 18:02:30 +0100 (CET)

Coucou Jon,
Je veux des lasagnes :)
Bye

.
QUIT
+OK Logging out.
Connection closed by foreign host.
```

Question :

Implémentez un client POP3 qui permet de lire un mail étant donné un numéro. Notez que l'adresse du serveur POP3, son port, le nom de l'utilisateur, le mot de passe ainsi que le numéro du mail à lire font partie des paramètres d'entrée de ce client.