EXAMEN PROGRAMMATION RÉSEAU 17/05/2016 15^{30}-18^{30}

Les seuls documents autorisés sont les documents « sur papier » ;
à l'exclusion de la copie ou des brouillons des voisins.

Consignes

- Lire **très attentivement** tout le sujet.
- Il est tout à fait possible de répondre à une question sans avoir répondu à toutes les questions précédentes et aussi d'utiliser les méthodes des questions précédentes même si on n'a pas donné leur code.
- Pour répondre aux questions, pensez à utiliser le code des réponses précédentes.
- À la fin du document, on rappelle quelques méthodes `Java` et `C` qui pourraient être utiles.
- Quand des consignes manquent de précision (par exemple le type de retour des méthodes), libre à vous de proposer la solution qui vous semble le mieux.
- On supposera que tous les messages circulant sur le réseau sont corrects (on ne vous demande pas de traiter les cas où un message ne respecte pas la spécification).
- Les messages circulant sont signalés entre crochets `[...]` et **les crochets ne font pas partie du message.**

Contexte

On souhaite développer un service de communication qui reçoit des messages à envoyer à un identifiant, se connecte à un annuaire pour connaître les adresses IP associées à l'identifiant et envoie ensuite le message correspondant. Nous aurons à notre disposition également un contrôleur qui peut tester si une machine souhaitant envoyer un message n'est pas blacklistée. Ainsi notre service sera client de l'annuaire et du blacklister.
Vous devez seulement programmer le service et supposer que l'annuaire et le blacklister existent déjà.

L'annuaire

On dispose sur la machine `annuaire.pr16.fr` d'un annuaire associant des identifiants (qui sont des chaînes de 8 caractères alphanumériques) à des adresses IP. On supposera que la taille d'une chaîne de caractères associée à une adresse IP est au plus de 15 caractères et que les adresses IP sont des adresses IPv4. Un identifiant peut avoir plusieurs adresses IP. Ces adresses IP correspondent à des machines qui ont un client attendant des messages sur leur port **UDP** 8888. La communication avec l'annuaire se fait sur le port **TCP** 7777. Une fois qu'un client de l'annuaire s'est connecté, il peut faire des requêtes auxquelles l'annuaire répond et le client choisit quand fermer la connexion. Plus précisément, voilà les requêtes que le client peut envoyer et les réponses attendues

- Le client peut demander une adresse IP correspondant à un identifiant en envoyant une requête `[U id\n]` où `id` est une chaîne de caractères correspondant à l'identifiant. L'annuaire répondra alors soit par un message `[E\n]` si l'identifiant n'a pas d'IP correspondant, soit par un message `[I ip\n]` où `ip` est une chaîne de caractères contenant une adresse IP associée à l'identifiant.
- Le client peut demander toutes les adresses IP correspondant à un identifiant en envoyant une requête `[A id\n]`. L'annuaire répondra alors par un premier message `[N n\n]` où `n` est une chaîne de caractère contenant un entier positif ou nul indiquant le nombre d'adresses IP correspondant à ce message, puis il enverra n messages de la forme `[L ip\n]`, chacun de ces messages contenant une chaîne de caractères contenant une adresse IP associée à l'identifiant.

Un client de l'annuaire peut faire plusieurs requêtes à la suite et c'est lui qui met fin à la communication en se déconnectant.

Le blacklister

Comme dit précédemment, nous disposons également sur la machine `blacklister.pr16.fr` d'un service qui écoute en **TCP** sur le port 6666 et qui permet de tester si l'adresse d'une machine correspond à une machine malsaine que l'on n'autorise pas à envoyer des messages. La communication avec ce service se fait de la façon suivante : un client se connecte et envoie un message de la forme `[T ip\n]` où `ip` est une chaîne de caractères contenant une adresse IP, ce à quoi le service répond par un message `[G\n]` si l'adresse IP n'est pas malsaine soit `[M\n]` si elle est malsaine. Le service ferme ensuite la connexion.

Notre service d'envoi de message

Le service que nous souhaitons développer aura les fonctionnalités suivantes, il attend des connexions **TCP** sur son port 5555 qui sont des requêtes pour envoi de messages à des identifiants. Quand un client se connecte, le service commence par lui envoyer un message `[HI\n]`, il attend ensuite une requête. Après avoir reçu une requête il envoie un message `[BYE\n]` pour signaler qu'il a traité la requête (attention au fait que traiter la requête ne signifie pas forcément envoyer un message, par exemple si l'identifiant n'a pas d'IP, le service n'envoie pas de message, mais il envoie `[BYE\n]` quand même à son client). Il y a deux types de requêtes que des clients de notre service peuvent envoyer :

1. La requête `[S id mess\n]` est une requête d'envoi du message `mess` (chaîne de caractères d'au plus 100 caractères sans caractère `\n` ou `\r`) à une adresse IP correspondant à l'identifiant `id`. Quand il reçoit cette requête, notre service récupère auprès de l'annuaire une adresse IP de l'identifiant et envoie le message `mess` à l'identifiant. Nous supposons que les machines des identifiants attendent des messages d'au plus 101 caractères se terminant par `\n`. Si l'identifiant n'a pas d'adresse IP, notre service n'envoie rien.
2. La requête `[D id mess\n]` est une requête d'envoi du message `mess` (chaîne de caractères d'au plus 100 caractères sans caractère `\n` ou `\r`) à **toutes** les adresses IP correspondant à l'identifiant `id`. Quand il reçoit cette requête, notre service récupère auprès de l'annuaire les adresses IP de l'identifiant et envoie pour chacune d'elles le message `mess`.

Bien entendu, si un identifiant n'a pas d'adresse IP associée dans l'annuaire, notre service n'envoie pas le message. Notre service aura de plus un mode `safe`, si il est dans ce mode, avant d'envoyer un message il testera au préalable auprès du blacklister si l'adresse IP du client est malsaine ou non, et dans le cas où elle est malsaine il n'enverra pas les messages demandés. Attention, l'adresse IP testée n'est pas celle de l'identifiant mais celui du client demandant l'envoi de messages.

Questions

Nous créons tout d'abord une classe de base contenant les différentes fonctionnalités qui nous seront utiles pour programmer notre service. Cette classe s'appelle `CommService` et elle ne dispose que d'un attribut booléen `safe` qui est vrai si notre service est en mode `safe` et faux sinon. Notre but est dans un premier temps de compléter le code de cette classe.

```
public class CommService{
  public boolean safe;

  public CommService(boolean _safe){
    this.safe = _safe;
  }
}
```

1. Écrire une méthode `envoieMess` qui prend en argument une adresse IP (correspondant à un identifiant) sous forme de chaîne de caractères et un message sous forme de chaîne de caractères et envoie le message à l'adresse IP. Bien entendu, ce message sera envoyé en UPD au port précisé dans l'énoncé.

2. Écrire une méthode `envoieMultMess` qui prend en argument une liste de type `LinkedList<String>` d'adresses IP (correspondant à un identifiant) et un message sous forme de chaîne de caractères et envoie le message à toutes les adresses IP. La liste peut être vide.
3. Écrire une méthode `recupIP` qui prend en arguments une socket TCP connectée à l'annuaire et une chaîne de caractères correspondant à un identifiant et renvoie la chaîne de caractères contenant l'IP de cet identifiant obtenu auprès de l'annuaire (si il n'y a pas d'IP, la méthode renvoie `null`).
4. Écrire une méthode `recupAllIP` qui prend en arguments une socket TCP connectée à l'annuaire et une chaîne de caractères correspondant à un identifiant et renvoie une liste de type `LinkedList<String>` contenant toutes les chaînes de caractères correspondant aux adresses IP de cet identifiant obtenues auprès de l'annuaire (si il n'y a pas d'IP, la méthode renvoie une liste vide).
5. Écrire une méthode `estOk` qui prend en argument une socket TCP (de type `Socket`) et teste auprès du blacklister si l'adresse IP correspondante à la socket n'est pas malsaine. Cette méthode renvoie un booléen égal à vrai si l'adresse n'est pas malsaine et à faux sinon.
6. Écrire une méthode `traiteReq` qui prend en argument une chaîne de caractères correspondant à une requête d'un client de notre service (messages commençant par `S` ou `D`) et la traite comme expliqué auparavant. On ne teste pas dans cette méthode si la requête vient d'une machine malsaine ou non. Cette méthode devra entre autre se connecter à l'annuaire.
7. Écrire une méthode `traiteSafeReq` qui prend deux arguments une socket (celle avec laquelle notre service communique avec le client) et une chaîne de caractères correspondant à une requête et si notre service est en mode `safe`, teste si l'adresse IP du client est malsaine et dans le cas négatif traite la requête et si notre service n'est pas en mode `safe` traite la requête sans faire de test.

On va maintenant décrire la classe `TraiteConnexion` qui s'occupera de traiter chaque connexion avec les différents clients de notre service. Cette classe aura deux attributs, le premier de type `CommService` et le deuxième de type `Socket`. Dans notre service, à chaque connexion on créera un nouvel objet `TraiteConnexion` dont la fonction `run` aura pour rôle de traiter la connexion faite via l'objet `Socket` correspondant. Tous les objets `TraiteConnexion` du programme partageront le même objet `CommService`.

```
public class TraiteConnexion implements Runnable{
  public Socket soc;
  public CommService cs;

  public TraiteConnexion (Socket _soc,CommService _cs){
    this.soc = _soc;
    this.cs = _cs;
  }
}
```

8. Écrire la méthode `run` de la classe `TraiteConnexion`. Cette méthode sera appelée après une connexion au service et elle sera responsable de la communication entre le client et notre service.
9. Écrire une classe `ProgPrincipal` qui contiendra le programme principal (c'est-à-dire le `main`) de notre service. Ce programme créera tout d'abord un objet `CommService` qui sera `safe` si un argument (n'importe lequel) est donné au programme et sinon il ne sera pas `safe`. Notre programme attendra ensuite en boucle des connexions sur le port de notre service et lancera pour chaque connexion un thread pour gérer la communication.

On va maintenant écrire différents clients en C.

10. Écrire un client en C pour notre service en supposant que celui-ci tourne sur la machine `machine.pr16.fr`. Notre client demandera à envoyer le message `Hello World!` à une adresse IP de l'identifiant `A1B2C3D4`.

11. Écrire un client en C pour l'annuaire qui affichera toutes les adresses IP associés à l'identifiant `A1B2C3D4`. Il pourra afficher les messages tels quels contenant les adresses IP renvoyés par l'annuaire.
12. Écrire en C le code d'un programme associé à un identifiant enregistré sur l'annuaire et qui attend des messages et les affiche.

On souhaite améliorer notre service pour limiter les connexions avec l'annuaire. Pour cela, notre service enregistrera les requêtes dans une liste de chaînes de caractères au lieu de les traiter directement et quand cette liste contiendra 20 éléments, il traitera les requêtes. Pour ce faire on développera la classe `CommServiceAmeliore` où le champ `nombreReq` compte le nombre de requêtes dans la liste.

```
public class CommServiceAmeliore extends CommService{
  public LinkedList<String> reqLi;
  public int nombreReq;

}
```

13. Écrire le constructeur de la classe `CommServiceAmeliore`.
14. Redéfinir la méthode `traiteReq` pour qu'elle prenne maintenant en compte le nouveau comportement de notre service. Attention aux accès concurrents à la liste de requêtes.
15. Écrire une classe `ProgPrincipal2` qui contiendra le programme principal (c'est-à-dire le `main`) de notre service amélioré et qui fonctionnera de façon similaire à `ProgPrincipal` pour la gestion du mode `safe`.

Quelques fonctions de Java pouvant être utiles

— Méthodes de la classe `Socket`

```
InetAddress getInetAddress()
\\ Returns the address to which the socket is connected.
```

— Méthodes de la classe `InetAddress`

```
static InetAddress getByName(String host)

String getHostAddress()
//Returns the IP address string in textual presentation.
```

— Méthodes de la classe `InetSocketAddress`

```
public InetSocketAddress(String hostname, int port)
//hostname is either the name of the machine or the IP address
```

— Méthodes de la classe `LinkedList<E>`

```
boolean add(E e)

ListIterator<E> listIterator()
```

— Méthodes de la classe `ListIterator<E>`

```
boolean hasNext()
```

```
E next()

void remove()
/*Removes from the list the last element that was returned by next()
or previous() (optional operation). This call can only be made once
 per call to next or previous. It can be made only if add(E) has not
been called after the last call to next or previous.*/
```

— Méthodes de la classe String

```
int length()

String substring(int beginIndex, int endIndex)
//Returns a new string that is a substring of this string.
// The substring begins at the specified beginIndex
// and extends to the character at index endIndex - 1.
// Thus the length of the substring is endIndex-beginIndex.
// The first index is 0.
```

— Méthodes de la classe Integer

```
Integer(String s)
/*Constructs a newly allocated Integer object that represents the int
value  indicated by the String parameter. */

int intValue()
```

Quelques fonctions de C pouvant être utiles

```
int atoi(const char *str);
/*convert ASCII string to integer*/
```