EXAMEN | PROGRAMMATION RÉSEAU | 7/5/2018 8^{30}-11^{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` 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**.
- On supposera qu'un message envoyé dans un paquet UDP est reçu dans un paquet UDP (c'est-à-dire qu'il n'est pas coupé).
- On ne demande pas de traiter les exceptions et on ne demande pas d'écrire les `import` d'API

Contexte

Le but de ce sujet est de programmer un protocole de communication basé sur une communication en anneau. L'idée

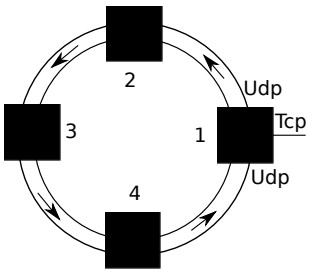

FIGURE 1 – Schéma d'un anneau

est que des entités communiquent entre elles par UDP et la topologie du réseau est un anneau. La circulation des messages sur un anneau se fait de façon unidirectionnelle. Ainsi sur l'exemple d'anneau représenté par la Figure 1, l'entité 1 envoie ses messages sur le port UDP d'écoute de l'entité 2, l'entité 2 envoie ses messages sur le port UDP d'écoute de l'entité 3, etc. Chaque entité peut de plus attendre une connexion sur un port TCP pour permettre l'insertion d'un nouveau noeud dans l'anneau ou la demande d'émission de messages dans l'anneau.

Le but sera de programmer des entités pour réaliser ce protocole de communication en anneau et de programmer des clients utilisant cet anneau.

Les entités

Les caractéristiques d'une entité sont les suivantes : un port d'écoute pour recevoir les messages UDP de l'entité précédente sur l'anneau (inférieur à 9999), un port TCP, l'adresse IP de l'entité suivante sur l'anneau et le port d'écoute UDP de l'entité suivante sur l'anneau. Nous supposerons que deux entités n'ont pas de couples adresse IP-port TCP ou adresse IP-port UDP identiques.

Insertion dans un anneau

Nous allons maintenant décrire comment se déroule l'insertion d'une nouvelle entité dans un anneau. Supposons par exemple qu'une entité souhaite s'insérer dans l'anneau décrit à la Figure 1 entre l'entité 1 et l'entité 2. Pour se faire elle commencera par se connecter au port TCP de l'entité 1, celle-ci lui enverra alors le message suivant [`WELC␣ip␣port\n`] où `ip` et `port` sont les caractéristiques de l'entité 2 sur laquelle la nouvelle entité devra transmettre ces messages. La nouvelle entité répondra alors avec un message de la forme [`NEWC␣port\n`] avec son numéro de port d'écoute UDP. Suite à quoi, l'entité 1 répondra par le message [`ACKC\n`] signalant qu'à partir de maintenant elle enverra à cette nouvelle entité les messages UDP à transmettre, puis elle fermera la connexion TCP. À partir du moment où elle envoie le message [`ACKC\n`], l'entité 1 enverra désormais les messages UDP à transmettre à la nouvelle entité et non plus à l'entité 2.

Messages circulant sur l'anneau

Sur l'anneau circule des messages en UDP. Afin que les messages circulant sur l'anneau ne tournent pas indéfiniment chaque message aura un identifiant `idm` spécifique et **unique** encodé sur 4 octets et toute entité recevant un message déjà transmis ne devra pas le retransmettre à son successeur sur l'anneau.
Les messages circulant sur l'anneau ont la forme suivante [`MESS␣idm␣message`] où `idm` est l'identifiant unique du message circulant sur l'anneau et `message` est le message. Quand une entité sur l'anneau voit un message elle le retransmet à l'entité suivante sauf si elle a déjà vu ce message ou si elle a elle-même émis ce message.

Les clients

Les clients peuvent se connecter à une entité sur l'anneau pour lui demander d'émettre un message sur l'anneau. Un client se connect alors en TCP sur le port correspondant de l'anneau, il reçoit le message [`WELC␣ip␣port\n`] qu'il ne prend pas en compte et il envoie à l'entité le message [`MES!␣message\n`] et se déconnecte. Lorsqu'une entité reçoit un tel message, elle crée un identifiant unique pour le message et elle l'envoie sur l'anneau comme présenté précédemment.

Format

Voilà le format des différents champs des messages :

- `ip` est une chaîne de caractères d'au plus 15 caractères contenant la représentation d'une adresse IP
- `port` est une chaîne de caractères d'au plus 4 caractères contenant la représentation textuelle d'un numéro de port. Par exemple, le numéro 15 sera représenté par la chaîne `"15"`.
- `idm` est un chiffre encodé sur 4 octets.
- `message` est une chaîne de caractères d'au plus 300 caractères contenant un message.

Questions

Le but de cet examen est de programmer les entités ainsi que des clients.

La première étape consiste à compléter la classe `Entity` dont nous donnons le squelette ci-dessous.

```
public class Entity {
  /*Port TCP*/
  public int port_tcp;

  /*Port UDP*/
  public int port_udp;

  /*IP UDP du suivant*/
  public String next_ip;

  /*Port UDP du suivant*/
  public int next_port_udp;

  /*Liste des identifiants des messages emis ou lus*/
  public LinkedList<byte[]> listMess;
}
```

1. Écrire un constructeur de `Entity` qui prend comme arguments un entier correspondant au port d'écoute TCP et un entier correspondant au port UDP et remplit les champs correspondants. On supposera qu'avec ce constructeur l'entité suivante est elle-même, on forme ainsi un anneau d'une entité. Ainsi l'adresse IP de l'entité suivante et son port UDP sont les mêmes que l'entité appelante. La liste sera aussi initialisée par ce constructeur.

Nous allons commencer par programmer la partie liée à la circulation des messages UDP.

2. Écrire une methode `inList` de la classe `Entity` qui prend en argument un tableau d'octets contenant un message UDP circulant sur l'anneau, extrait l'identité unique du message et teste si elle est présente dans la liste `listMess`. Cette méthode renverra un booléen correspondant au test. De plus si l'identité n'est pas présente dans la liste, cette méthode l'ajoute. On supposera que les caractères `MESS` au début des messages occupent chacun un octet.
3. Écrire une méthode `sendUDP` de la classe `Entity` qui prend en argument un tableau d'octets contenant un message UDP circulant sur l'anneau et le transmet à l'entité suivante. Cette méthode ne testera pas si l'identité du message est présent dans la liste et ouvrira et fermera les sockets nécessaires à cette communication.
4. Écrire une méthode `copyByte` de la classe `Entity` qui prend comme arguments un tableau d'octets et un entier correspondant à une longueur l et renvoie un tableau d'octets contenant une copie des l premières cases du tableau donné en entrèe.

Comme les entités devront à la fois traiter des messages TCP et des messages UDP, elles seront composées de deux threads. La classe `EntityUDP` sera responsable du thread gérant les messages UDP.

```
public class EntityUDP implements Runnable {
  Entity ent;

  public EntityUDP(Entity _ent){
    ent = _ent;
  }
}
```

5. Écrire la méthode `run` de la classe `EntityUDP` qui créera la socket d'écoute UDP et en boucle attendra les messages UDP et les traitera en ne transmettant pas les messages émis ou déjà vus par l'entité.

.
On peut maintenant passer à la partie TCP.

6. Écrire une méthode `sendSucc` de la classe `Entity` qui prend comme argument un objet de la classe `PrintWriter` d'une socket TCP déjà connectée et envoie un message sur cette socket de la forme [`WELC␣ip␣port\n`] avec les informations de l'entité suivante sur l'anneau.
7. Écrire une méthode `treatNew` de la classe `Entity` qui prend comme arguments une chaîne de caractères contenant une adresse IP et une chaîne de caractères contenant un message de la forme [`NEWC␣port\n`] et qui attribue à l'entité ces deux éléments comme les nouveaux successeurs dans l'anneau.
8. Écrire une méthode `treatMess` de la classe `Entity` qui prend comme argument une chaîne de caractères contenant un message de la forme [`MES!␣message\n`], qui génère un nouvel identifiant de message, l'enregistre dans la liste et envoie le message de type [`MESS`] correspondant sur l'anneau. Pour générer un nouvel identifiant, on supposera l'existence d'une méthode statique `byte[] Util.magicNumber()` qui génère un tableau de 4 octets contenant un chiffre unique et ce peu importe d'où l'appel est fait.

Nous pouvons maintenant passer à la classe `EntityTCP` qui sera responsable de l'activité du thread gérant les activités TCP. Notons que dans notre implémentation, comme les échanges TCP sont brefs nous n'autoriserons pas plusieurs connexions TCP en parallèle.

```
public class EntityTCP implements Runnable {
  Entity ent;

  public EntityTCP(Entity _ent){
    ent = _ent;
  }
}
```

9. Écrire la méthode `run` de la classe `EntityTCP` qui en boucle attend une connexion TCP, envoie le message de type [`WELC`], attend un message TCP et le traite selon son type ([`NEWC`] ou [`MES!`]) en réalisant l'action correspondante. N'oubliez pas que cette méthode devra dans le cas des messages d'insertion dans l'anneau aussi envoyer le message de type[`ACKC`].
10. Écrire la méthode `launch` de la classe `Entity` qui crée les deux threads responsables de gérer les communications TCP et UDP.
11. Les deux threads gérant les communications UDP et TCP accèdent tous les deux de façon concurrente à la liste des identifiants de message. Expliquer de façon précise ce qu'il faudrait faire pour que cette accès se passe sans conflit.
12. Il faut maintenant écrire les méthodes de la classe `Entity` permettant de se connecter à un anneau. Écrire une méthode `connectRing` de la classe `Entity` qui prend comme arguments une chaîne de caractères contenant une adresse IP d'une machine où tourne une entité et un entier correspondant au numéro de port TCP de cette entité, qui se connecte et fait les opérations pour s'insérer dans l'anneau après l'entité à laquelle elle s'est connectée. *Indication :* On pourra utiliser la méthode `String[] split(String regexp)` de la classe `String` qui si on lui donne comme arguments `"␣"` découpe la chaîne appelante selon les espaces.
13. Dans une classe `MainProg1`, écrire un programme qui crée une entité et lance son exécution.
14. On suppose que le programme précédent s'exécute sur la machine `intothering.examenpr.fr`, écrire dans une classe `MainProg2` un programme qui crée une entité, la fait s'insérer dans l' anneau de taille 1 créé à la question précédente et lance son exécution.

Passons maintenant à une partie en C.

15. Écrire en C, un client pour l'entité de la question 13 s'exécutant sur la machine `intothering.examenpr.fr`. qui lui demande d'envoyer le message `"Hello World!"` sur l'anneau.

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 getLocalHost()
// Gets the local Address
//to the specified OutputStream.

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

static InetAddress getByName(String host)
//Determines the InetAddress of a host, given the host's name or IP.
```

— Méthodes de la classe `LinkedList<E>`

```
boolean add(E e)

ListIterator<E> listIterator()
```

— Méthodes de la classe `ListIterator<E>`

```
boolean hasNext()

E next()
```

— 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.

static String valueOf(int i)
//Returns the string representation of the int argument.
```

— Méthodes de la classe `Integer`

```
static int parseInt(Sting s)
/*Parses the string argument as a signed decimal integer*/
```

— Méthode de la classe `Array`

```
static boolean  equals(byte[] a, byte[] a2)
/*Returns true if the two specified arrays are equal to one another.*/
```