EXAMEN | PROGRAMMATION RÉSEAU | 15/05/2017 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` 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 ne demande pas de traiter les exceptions et on ne demande pas d'écrire les `import` d'API

Contexte

Nous disposons de serveurs d'images qui acceptent des connexions en TCP pour transmettre des images qu'ils ont. Le but de ce sujet est de programmer un méga-serveur auprès duquel les serveurs d'images s'enregistreront et sur lequel les clients se connecteront pour lui demander de rechercher une image (donnée par son nom) et de la transmettre. Dans un premier temps, nous expliquerons comment les serveurs d'images fonctionnent ensuite nous verrons comment programmer le méga serveur.

Les serveurs d'images

Un serveur d'images est caractérisé par une adresse IP et par un port de connexion TCP. Nous supposerons que deux serveurs n'ont pas de couple adresse IP-port TCP identique. Pour récupérer une image de nom `nom-image` sur un serveur d'images, on se connecte au serveur sur le port TCP correspondant et on envoie le message `[I nom-image\n]`, si le serveur n'a pas cette image il répond `[N\n]` et si en revanche il l'a, il répond en envoyant d'abord le message `[O\n]` puis en envoyant l'objet de la classe `MyImage` sérialisé. Le serveur d'images ferme ensuite la connexion.

Le méga-serveur

Les clients de notre application se connectent au méga-serveur qui se connecte lui aux serveurs d'images pour répondre aux requêtes des clients. De plus, les serveurs d'images peuvent s'enregistrer auprès du méga-serveur. Le méga-serveur est caractérisé par un port UDP utilisé par les serveurs d'image pour s'enregistrer et un port TCP utilisé par les clients.
L'idée est la suivante, lorsqu'un client souhaite une image `nom-image` il envoie un message `[R nom-image\n]`au méga-serveur qui interroge alors tous les serveurs d'images qui se sont enregistrés auprès de lui jusqu'à trouver l'image. Si il la trouve, il répond au client `[V\n]` et il lui transmet ensuite l'image sérialisée, et sinon il répond au client `[K\n]`. Il ferme ensuite la connexion. Pour tester si un serveur d'images a une image il la lui demande comme vu précédemment.
Pour s'enregistrer auprès du méga-serveur, un serveur d'images envoie simplement sur le port UDP du méga-serveur un message `[R port]` où `port` est son numéro d'écoute TCP qui est encodé par une chaîne de 4 caractères correspondant

au numéro de port (ainsi un serveur d'images est contraint d'avoir un numéro de port TCP compris entre 1000 et 9999). Le méga serveur ne répond rien à ce message, si le serveur d'images n'est pas dans sa liste il l'ajoute et sinon il ne fait rien. De la même façon un serveur d'images peut se désenregistrer en envoyant sur le port UDP du serveur un message `[D port]` qui a la même forme que le message d'enregistrement. Dans ce cas, si le serveur d'images est présent dans la liste du méga-serveur celui-ci le retire et sinon il ne fait rien. *Remarque :* Il est normal que l'adresse IP du serveur d'images ne figure pas dans le message car le méga-serveur a la possibilité de la récupérer.

Le but de cet examen est de programmer le méga-serveur ainsi que des clients.

Questions

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

```
/**
 * classe permettant de gerer des images
 */
public class MyImage implements Serializable{

    public int[][] rgb;

    /**
     * construit une image a partir d'un fichier image (jpeg/png/bmp/wbmp/gif)
     * @param path le fichier image
     * @throws IOException en cas d'erreur d'entr&eacute;ee/sortie
     */
    public MyImage(String path) throws IOException {
        BufferedImage bufIm = ImageIO.read(new File(path));
        rgb = new int[bufIm.getWidth()][bufIm.getHeight()];
        for(int i=0; i<bufIm.getWidth(); i++)
            for(int j=0; j<bufIm.getHeight(); j++){
                rgb[i][j] = bufIm.getRGB(i, j);
            }
    }

    /** @return largeur de l'image */
    public int getWidth(){
        return rgb.length;
    }

    /** @return hauteur de l'image */
    public int getHeight(){
        return rgb[0].length;
    }

    /**
     * @return une instance de BufferedImage representant l'image
     */
    public BufferedImage toBufferedImage(){
        BufferedImage res = new BufferedImage(getWidth(),
            getHeight(), BufferedImage.TYPE_INT_RGB);
```

```
        for(int i=0; i<getWidth(); i++)
            for(int j=0; j<getHeight(); j++)
                res.setRGB(i, j, rgb[i][j]);
        return res;
    }
}
```

1. Écrire une méthode `writeFile` de la classe `myImage` qui prend en argument un nom de fichier et écrit l'image correspondant dans le fichier au format `jpeg`. Pour cela vous pourrez utiliser la méthode `static boolean write(BufferedImage im, String formatName, File output)throws IOException` de la classe `ImageIO` où `formatName` vaudra `"jpeg"`.

Nous passons maintenant à la programmation de la classe `MegaServer` et des classes qu'elle utilisera. Tout d'abord nous programmerons la classe `ImageServer` qui correspondra à un serveur d'images enregistré dans le méga-serveur. Le squelette de cette classe est le suivant :

```
public class ImageServer {
   /*IP du serveur d'images*/
   public String Ip;

   /*Port TCP du serveur d'images*/
   public int port;
}
```

2. Écrire un constructeur de `ImageServer` qui prend comme arguments une chaîne de caractères correspondant à une adresse IP et une chaîne de caractères de taille 4 correspondant à un numéro de port et qui remplit les deux champs `Ip` et `port`.
3. Écrire une méthode `connect` de la classe `ImageServer` qui se connecte au serveur d'images et renvoie la socket correspondante à cette connexion. Cette méthode ne prend pas d'arguments.
4. Écrire une méthode `request` de la classe `ImageServer` qui prend en arguments une socket TCP connectée au serveur d'images et un nom d'image `nomImage` et envoie une requête au serveur d'images pour demander l'image.
5. Écrire une méthode `getAnswer` de la classe `ImageServer` qui renvoie un objet de la classe `MyImage` et prend en arguments une socket TCP connectée au serveur d'images à qui une requête a été envoyée, attend les réponses et si la réponse est `[0\n]` attend l'image sérialisée et retourne l'objet `MyImage` correspondant. Dans le cas d'une réponse négative cette méthode renvoie `null`. (*Indication :* Vous pourrez d'abord utiliser un *BufferedReader* pour lire le message et ensuite un *ObjectInputStream* pour lire l'objet).
6. Écrire une méthode `getImage` de la classe `ImageServer` qui prend comme arguments un nom d'images et qui se connecte au serveur d'images, fait une requête d'images et si l'image est envoyée retourne l'objet `MyImage` correspondant et sinon retourne `null`.

Nous allons maintenant nous attaquer à la classe `MegaServer` dont un squelette de code est donné ci-après.

```
public class MegaServer{
   /*port TCP du MegaServer*/
   public int portTCP;

  /*port UDP du MegaServer*/
  public int portUDP;
```

```
  /*Liste des Server d'image*/
  public LinkedList<ImageServer> listImageServ;
}
```

7. Écrire un constructeur de la classe `MegaServer` qui prend en arguments deux ports entiers, initialise les champs correspondants et crée la liste `listImageServ`.
8. Écrire une méthode `addImageServ` de la classe `MegaServer` qui prend en arguments une chaîne de caractères correspondant à l'adresse IP d'un serveur d'images et une chaîne de caractères correspondant à son numéro de port et qui ajoute l'objet de la classe `ImageServer` à la liste (un serveur d'images peut être ajouté plusieurs fois).
9. Écrire une méthode `removeImageServ` de la classe `MegaServ` qui prend en arguments une chaîne de caractères correspondant à l'adresse IP d'un serveur d'images et une chaîne de caractères correspondant à son numéro de port et qui retire le serveur d'images correspondant de la liste si il est dedans et sinon ne fait rien (si un serveur d'images est présent plusieurs fois on les retire tous).

La fonction principale de la classe `MegaServer` va devoir lancer plusieurs types de threads, un pour gérer les messages arrivant sur le port UDP correspondant à l'enregistrement et au désenregistrement des serveurs d'images, l'autre gérant les différentes communications avec les clients. Commençons d'abord par écrire le code de la classe qui gérera les communications UDP. On considère la classe `TreatUDP`.

```
public class TreatUDP implements Runnable{
   /*Mega Serveur associe*/
   MegaServer ms;

   /*Constructeur*/
   public TreatUDP (MegaServer ms){
     this.ms = ms;
   }
}
```

10. Écrire une méthode `treatUDPMess` de la classe `TreatUDP` qui prend en arguments une chaîne de caractères correspondant à un message UDP de la forme `[R port]` ou `[D port]` ainsi qu'une chaîne de caractères contenant l'adresse IP du serveur d'images ayant envoyé ce message et qui effectue l'action correspondante au message sur le `MegaServer ms`.
11. Écrire la méthode `run` de la classe `TreatUDP` qui crée une socket UDP pour attendre des messages sur le port UDP du Mega-Serveur `ms` et traite en boucle chaque message en récupérant l'adresse IP d'où il a été envoyé et en utilisant la fonction `treatUDPMess`.

Nous allons maintenant passer à la classe `TreatTCP` responsable des communications TCP.

```
public class TreatTCP implements Runnable{
   /*Mega Serveur associe*/
   MegaServer ms;

   /*Socket TCP correspondante au client*/
   Socket so;

   /*Constructeur*/
   public TreatTCP (MegaServer ms, Socket so){
     this.ms = ms;
```

```
        this.so = so;
    }
}
```

12. Écrire une méthode `searchImage` de la classe `TreatTCP` qui prend en arguments un nom d'images `nomImage` et cherche parmi les serveurs de la liste du Mega-Serveur `ms` si l'un d'eux a l'image et si c'est le cas cette fonction renvoie l'objet `MyImage` contenant l'image correspondante et sinon la fonction renvoie `null` (Bien entendu cette fonction devra communiquer avec les serveurs d'image).
13. Écrire la méthode `treatTCPMess` de la classe `treatTCP` qui prend en arguments une chaîne de caractères de la forme `[I nom-image\n]` envoyé par le client et interroge les serveurs d'image à la recherche de `nom-image` et renvoie soit un objet `MyImage` correspondant à l'image trouvée, soit `null` dans le cas contraire.
14. Écrire la méthode `run` de la classe `TreatTCP` qui attend sur la socket TCP un message d'un client de la forme `[I nom-image\n]` et renvoie au client la réponse correspondant à sa requête selon si l'image a été trouvée ou non. Dans le cas où l'image a été trouvée, cette méthode la transmet au client. Cette méthode ferme ensuite la Socket et termine.

Nous retournons programmer la classe `MegaServer`.

16. Écrire la méthode `launch` de la classe `MegaServer` qui lance le thread d'écoute UDP puis attend en boucle des connexions sur son port TCP et pour chaque connexion lance un thread pour gérer la communication.
17. Écrire une méthode principale `main` qui crée un Mega serveur avec comme port UDP 4444 et comme port TCP 5555 et qui le fait s'exécuter.
18. Écrire dans une classe `Client` un client qui se connecte au serveur de la machine précédente en supposant que celui-ci s'exécute sur la machine `machine.anywhere.fr` et lui demande l'image `HeartShapedBox` et si le serveur l'a, notre client la sauvera dans le fichier `myprofile.jpeg`.
19. Écrire un client C qui demandera au serveur de la question 18 si les images `image1`, `image2`, ...,`image10` sont disponibles. Bien entendu ce client ne téléchargera pas les images correspondantes car il est écrit en C et ne peut donc pas gérer les images sérialisées, mais il attendra juste la réponse à ses requêtes qui sera de type `V` ou `K`.

Quelques fonctions de Java pouvant être utiles

— Méthodes de la classe `DatagramPacket`

```
InetAddress getAddress()
// Returns the address to which the socket is connected.
```

— Méthodes de la classe `InetAddress`

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

— Méthodes de la classe `LinkedList<E>`

```
boolean add(E e)

ListIterator<E> listIterator()
```

— Méthodes de la classe `ObjectInputStream`

```
ObjectInputStream(InputStream in)
// Creates an ObjectInputStream that reads from
// the specified InputStream.

Object  readObject()
//Read an object from the ObjectInputStream.
```

— Méthodes de la classe `ObjectOutputStream`

```
ObjectOutputStream(OutputStream out)
// Creates an ObjectOutputStream that writes
//to the specified OutputStream.

void writeObject(Object obj)
//Write the specified object to the ObjectOutputStream.

void flush()
//Flushes the stream.
```

— 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`

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