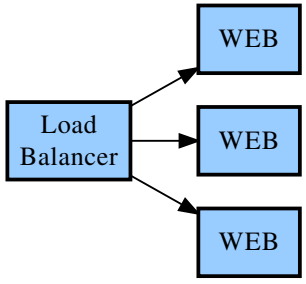
Intergiciels - Examen 2010-2011

Daniel Hagimont
Durée: 1 heures 45 minutes, documents autorisés
Lire l'ensemble des énoncés avant de commencer à répondre. La clarté, la précision et la concision des réponses, ainsi que leur présentation matérielle, seront des éléments importants d'appréciation. Donnez essentiellement les programmes Java demandés. Dans vos programmes, vous n'avez pas à programmer les imports et le traitement des exceptions.

PROBLÈME I (sockets)

Vous avez à réaliser avec des sockets en Java un répartiteur de charge (une classe LoadBalancer) qui reçoit des requêtes HTTP et les distribue de façon aléatoire vers un ensemble de serveurs Web. Le début de la classe LoadBalancer est le suivant :

```
public class LoadBalancer {
    static String hosts[] = {"host1", "host2"};
    static int ports[] = {8081,8082};
    static int nbHosts = 2;
    static Random rand = new Random();
}
```



A chaque réception d'une requête HTTP, LoadBalancer transfère la requête à un des serveurs web (les adresses de ces serveurs sont données par les tables hosts et ports) et LoadBalancer transfère le résultat de la requête à l'émetteur. Le choix du serveur Web est aléatoire (`rand.nextInt(nbHosts)` retourne un entier entre 0 et nbHosts-1). Pour être efficace, LoadBalancer est évidemment multi-threadé.

Pour la programmation des entrées/sorties, on utilisera :

```
InputStream
- public int read(byte[] b);
    // bloquante, retourne le nombre de bytes lus
OutputStream
- public void write(byte[] b, int off, int len);
    // écrit les len bytes à la position off
```

et on suppose que les requêtes et réponses sont lues ou écrites en un seul appel de méthode avec un buffer de 1024 bytes.

```
public class LB extends Thread {

    static String hosts[] = {"host1", "host2"};
    static int ports[] = {8081,8082};
    static int nbHosts = 2;
    static Random rand = new Random();
    Socket client;


    public LB(Socket s) {
        client = s;
    }

    public static void main(String arg[]) {
        ServerSocket ss = new ServerSocker(8080); // port 8080 par exemple
        while (true) {
            new LB(ss.accept()).start(); // /!\ start et pas pas de run sinon on rur
            // /!\ le thread principal du coup on bloque
        }
    }

    public void run() {
        int nb = LB.rand.nextInt(LB.nbHosts); // le serveur à qui on va filer la rec
        Socket server = new Socket(LB.hosts[nb], LB.ports[nb]);
        InputStream cis = client.getInputStream();
        InputStream sis = server.getInputStream();
        OutputStream cos = client.getOutputStream();
        OutputStream sos = server.getOutputStream();

        /* On fait l'hypothèse qu'on peut tout lire en une seule fois
         * (~ réel sur un réseau local). */
        int nbLus;
        byte[] buff = new byte[1024];
        nbLus = cis.read(buff);
        sos.write(buff, 0, nbLus);
        nbLus = sis.read(buff);
        cos.write(buff, 0, nbLus);
        server.close();
        client.close();

    }
}
```

PROBLÈME II (RMI)

Vous devez réaliser avec Java RMI une implantation d'un intergiciel de type JMS. Ce service est implanté sous la forme d'un serveur RMI (sur une machine serveur) qui implante l'interface `MOM`, qui fournit 2 méthodes ayant les prototypes suivants :

```
public void publish(String topic, Message message) // publie un message sur un topic
public void subscribe(String topic, CallBack cb) // s'abonne à un topic
```

L'abonnement (subscribe) à un topic prend en paramètre une référence à un objet d'interface CallBack permettant le rappel du client pour lui transmettre un message publié sur le topic. L'interface `CallBack` fournit la méthode de prototype :

```
public void onMessage(Message message)
```

Vous devez programmer ce service (interface `MOM` et classe `MOMImpl`) ainsi qu'un exemple de programme client démontrant l'utilisation du service lors d'une publication et d'un abonnement. Pour simplifier, je vous propose d'implanter une classe `MOMImpl` qui ne prend en compte qu'un unique topic de nom "topic".

```
public class MOM extends Remote {
    public void publish(String topic, Message message) throws RemoteException;
    public void subscribe(String topic, CallBack cb) throws RemoteException;
}
```

```
public class MOMImpl extends UnicastRemoteObject implements MOM {

    private Map<String, List<CallBack>> association; // association topics/abonnés

    public MOMImpl() throws RemoteException {
        association = new HashMap<>();
    }

    public void publish(String topic, Message message) throws RemoteException {
        for(CallBack subs : ) {
             try {
                subs.onMessage(message);
             } catch (Exception e) {
                 // ...
             }
         }
    }

    public void subscribe(String topic, CallBack cb) throws RemoteException {
        List<CallBack> l = association.get(topic);
        if (l==null) {
            l = new LinkedList<CallBack>();
            association.add(topic, l);
        }
        l.add(cb);
    }

    public static void main(String args[]) {
        LocateRegistry.createRegistry();
        Naming.bind("//localhost/MOM", new MOMImpl());
    }
}
```

Intergiciels - Examen 2011-2012

Daniel Hagimont
Durée: 1 heures 45 minutes, documents autorisés
Lire l'ensemble des énoncés avant de commencer à répondre. La clarté, la précision et la concision des réponses, ainsi que leur présentation matérielle, seront des éléments importants d'appréciation. Donnez essentiellement les programmes Java demandés. Dans vos programmes, vous n'avez pas à programmer les imports et le traitement des exceptions.

PROBLÈME I (sockets)

Losrque des abonnés à Internet sont connectés par la technologie ADSL (Asymmétric Digital Subscriber Line), le débit montant (upload) est nettement inférieur au débit descendant (download). Ceci implique que si UsagerClient télécharge un gros document donné par UsagerServer, le transfert est limité par le débit montant (upload) de UsagerServer. Pour diminuer le temps de transfert, il est possible pour UsagerClient de lancer le téléchargement de différents fragments du même document à partir de plusieurs usagers possédant une copie du document. Cette technique est utilisée par de nombreuses plate-formes de téléchargement dites P2P.

Nous voulons implanter le schéma de principe de cette technique. Vous devez programmer avec des sockets en Java deux programmes exécutés respectivement sur le client et sur les machines serveurs :

- UsagerClient télécharge des fragments d'un document depuis plusieurs serveurs
- UsagerServer_i fournit les fragments du document

Question 1

> Quelle propriété essentielle doit remplir UsagerClient pour que cette technique puisse bénéficier de l'existence de plusieurs serveurs pour accélérer le téléchargement?

Il doit etre multithreadé.

> Cette propriété est elle nécessaire pour les processus UsagerServer_i ?

Non. Pas forcement.

Question 2

On suppose que les programmes UsagerClient et UsagerServer_i ont les variables suivantes définies :

```
final static String hosts[] = {"host1", "host2", "host3"};
final static int ports[] = {8081,8082,8083};
final static int nb = 3;
static String document[] = new String[nb];
```

Les 3 premières variables donnent les adresses des serveurs. Un serveur démarre en prenant en paramètre son numéro de serveur (0,1 ou 2 , l'indice dans les tables hosts et ports). On ne gère qu'un seul document qui est ici un tableau de String (document) et on considère qu'un fragment est une String de ce tableau. On suppose que le tableau document est initialisé pour les serveurs (vous n'avez pas à le faire) et il doit être rempli par le téléchargement pour le client.

On simplifie donc ce schéma avec 1 document composé de 3 String, téléchargées depuis 3 serveurs.

Pour les requêtes, il est recommandé d'utiliser la sérialisation d'objets Java. Une requête peut simplement envoyer un numéro de fragment sur une connexion TCP et recevoir la String correspondant au fragment.

Donnez une implantation des programmes UsagerClient et UsagerServer_i en suivant ce schéma.

```
public class UsagerServer {

    final static String hosts[] = {"host1", "host2", "host3"};
    final static int ports[] = {8081,8082,8083};
    final static int nb = 3;
    static String document[] = new String[nb];

    public static void main(String arg[]) {
        // On récupère le numéro du serveur (ie le i de UsageServer_i)
        int me = Integer.parseInt(args[0]);
        ServerSocket ss = new ServerSocker(ports[me]);
        while (true) {
            socket s = ss.accept();
            ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
            ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
            int num = (int)ois.readObject();
            oos.writeObject(document[num]);
            s.close();
        }
    }
}
```

```
public class UsagerClient extends Thread {

    final static String hosts[] = {"host1", "host2", "host3"};
    final static int ports[] = {8081,8082,8083};
    final static int nb = 3;
    static String document[] = new String[nb];

    int me;
    public UsagerClient(int i) {
        me = i;
    }

    public static void main() {
        Thread t[] = new Thread[nb];
        for(int i=0; i < UsagerClient.nb; i++) {
            t[i] = new UsagerClient(i);
            t[i].start(); // et non pas .run !
        }

        for (int i = 0; i < nb; i++) {
            t[i].join();
        }
    }

    public void run() {
        Socket s = new Socket(hosts[me], ports[me]);
        ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
        ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
        oos.write(me);
        document[me] = (String)ois.readObject();
        s.close();
    }

}
```

PROBLÈME II (RMI)

Vous avez à réaliser un service d'exécution de commande à distance (comme rsh) en utilisant Java RMI.

Ce service est composé de 2 programmes :

- un programme Daemon.java qui doit être lancé sur toutes les machines vers lesquels on veut pouvoir lancer des commandes à distance. Ce programme enregistre un objet RMI sur son RMIRegistry local.
- Un programme RE.java qui permet l'exécution d'une commande à distance. Ce programme interagit avec le programme Daemon du site sur lequel il faut exécuter la commande.

Le programme RE s'utilise de la façon suivante : `java RE <host> <command>` .
Le programme RE reçoit dans `args[0]` et `args[1]` les 2 chaines de caractères host et command. Pour exécuter une commande localement sur les machines distantes, on utilise :

```
localExec(String cmd);
```

Question 3

Donnez une implantation en Java des programmes Daemon et RE.

```
public interface Daemon extends Remote {

    public void exec(String cmd) throws RemoteException;
}


}

public class DaemonImpl extends UnicastRemoteObject implements Daemon {
    public DaemonImpl() {} throws RemoteException;
    public void exec(String cmd) throws RemoteException {
        localExec(cmd);
    }

    public static void main(String args[]) {
        // pour ne pas avoir à lancer le RMIRegistry séparément
        LocateRegistry.createRegistry();
        Naming.bind("//localhost/Daemon", new DaemonImpl());
    }
}

public class RE {
    public static void main(String args[]) {
                    // \/ SURTOUT PAS DaemonImpl sinon on se fait défoncer
        Daemon d = (Daemon) Naming.lookup("//" + args[0] + '/Daemon');
        d.exec(args[1]);

    }
}
```

Question 4

On veut maintenant gérer l'affichage console de la commande exécutée à distance. On veut que les affichages de la commande soient effectués sur la console du client qui a lancé la commande.
On exécute maintenant une commande localement avec : `localExec(String command, Console console);`
Console est une interface Java qui fournit notamment la méthode `println(String s);`
La commande exécutée réalise ses affichages en appelant cette méthode sur l'objet Console passé en paramètre de `localExec()` .
Modifiez votre implantation précédente pour permettre l'affichage sur le poste client de la commande exécutée à distance.

```
public interface Console extends Remote {
    public void println(String l) throws RemoteException;
}

public class ConsoleImpl extends UnicastRemoteObject implements Console {

    public ConsoleImpl() {} throws RemoteException;

    public void println(String l) throws RemoteException {
        System.out.println(l);
    }
}
```

En plus, il faut ajouter aux méthose exec (interface+impl) & localExec le paramètre console et modifier localExec fait le travail pour appeler Console.println. Dans RE, on crée une ConsoleImpl lors de l’appel à d.exec()