

# Intergiciels

## Examen

Daniel Hagimont

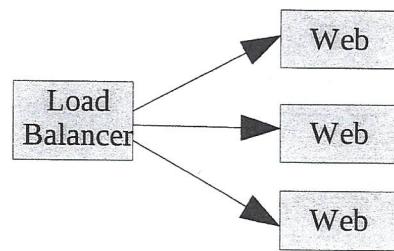
Durée: 1 heures 45 minutes, documents autorisés

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.

### 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".

# Intergiciels

Examen

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 (Asymétric 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 ?

Cette propriété est elle nécessaire pour les processus UsagerServer\_i ?

### 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.

## 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 chaînes 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.

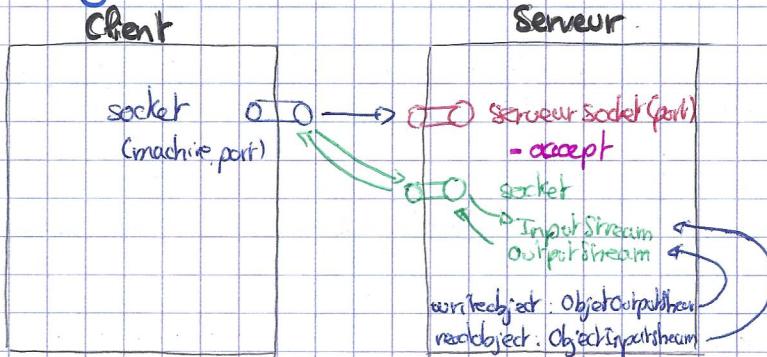
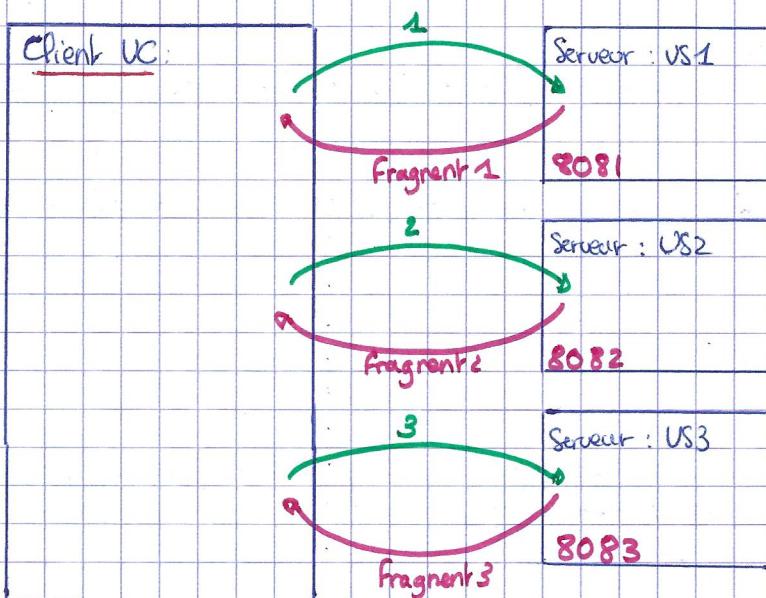
### Question 4

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.

Intergiciels TD 1Problème I, examen 2011-2012 : (sockets)

public class US; (Code pour le serveur i)

main (String args[]) {

try {

port = Integer.parseInt(args[0]);

ServerSocket ss = new ServerSocket(port); ← ouverture socket d'écoute

while (true) {

Socket s = ss.accept();

← ouverture socket de dialogue

InputStream is = s.getInputStream(); ← création d'InputStream (read)

OutputStream os = s.getOutputStream(); ← création d'OutputStream (write)

ObjectInputStream ois = new ObjectInputStream(is);

ObjectOutputStream oos = new ObjectOutputStream(os);

int frag = (int) ois.readObject();

oos.writeObject("voici le fragment " + frag + " de... "); ← On écrit sur l'objet à "envoyer"

close (is);

close (os);

close (s);

}

} catch (Exception e) {  
e.printStackTrace();

}.

;

Question 1: Pour le client il faut associer un thread pour chaque communication pour que le client puisse discuter avec plusieurs serveurs. Le serveur quant à lui peut être mono client.

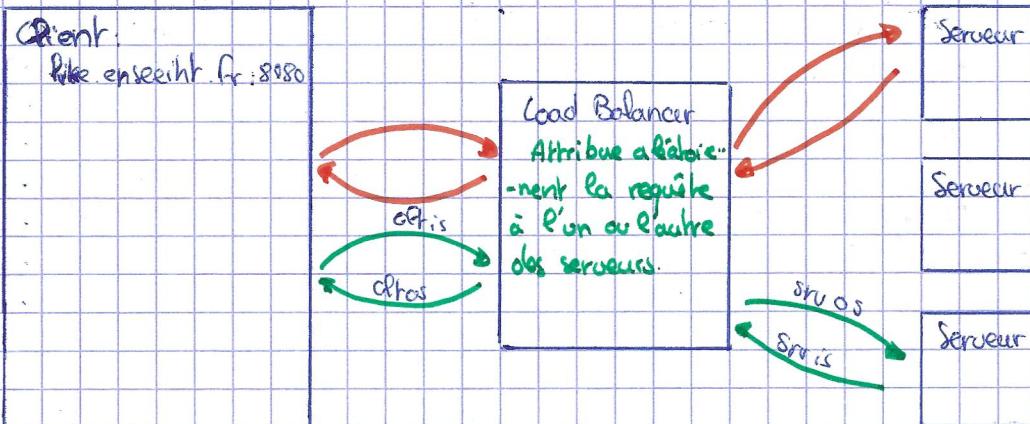
thread : processus léger, chargé d'exécuter un code particulier. Ainsi pour chaque thread créé correspondra une exécution du code différente.

```
public class UC { int num; static int port[] = { 8081, 8082, 8083 };  
main (String args[]) { static String document[] = new String[3];  
try { Thread th[] = new Thread[3];  
for (i=0; i<3; i++) {  
    th[i] = new UC(i);  
    th[i].start();  
}  
for (i=0; i<3; i++) {  
    th[i].join(); // on concatène les fichiers une fois qu'on les a tous reçus.  
}  
for (i=0; i<3; i++) {  
    System.out.println(document[i]); // Action simple pour vérifier ce qu'on a reçu depuis le serveur.  
} catch (Exception e) { e.printStackTrace(); }  
}  
}  
public UC (int i) { // Constructeur de l'UC  
    num = i;  
}  
public void run () { // Méthode automatique créée par la méthode start du thread  
    c'est cette méthode qui sera exécutée par le thread.  
    Socket s = new Socket ("localhost", port[num]);  
    InputStream is = s.getInputStream();  
    OutputStream os = s.getOutputStream();  
    ObjectInputStream ois = new ObjectInputStream (is);  
    ObjectOutputStream oos = new ObjectOutputStream (os);  
    oos.writeObject (num);  
    document [num] = (String) ois.readObject();  
    close (is);  
    close (os);  
    close (s);  
}  
}
```

Ouverture des Input / Output Stream comme pour le serveur

Fermeture des différents Stream et Socket.

## Problème I, examen 2010 - 2011 : (sockets)



```

public class LoadBalancer
    Socket cltsock;
main (String args [ ]) {
    ServerSocket ss = new ServerSocket (8080);
    while (true) {
        Thread t = new LB (ss.accept());
        ...
    }
}

public void run () {
    int i = rand.nextInt (nbPorts);
    Socket servSock = new Socket (Port[i]);
    srv is
    srv os
    clt is
    clt os
    byte [ ] buffer = new byte [1024];
    int readbytes;
    if (readbytes = clt.is.read (buffer) > 0) {
        srv.os.write (buffer, 0, readbytes);
        while (readbytes = srv.is.read (buffer) > 0) {
            clt.os.write (buffer, 0, readbytes);
        }
    }
}

```

## PROBLEME II (RHI) examen 2010-2011.:

Structure pour mémoriser les abonnés ?

→ Hashmap ayant pour clé les noms des topics et dont les valeurs sont des listes d'abonnés.

Quelle méthode utiliser ?

public Interface MOH extends Remote

public void publish (String topic, Message message) throws RemoteException  
public void subscribe (\_\_\_\_\_, Callback cb) \_\_\_\_\_

public class MORImpl extends UnicastRemoteObject implements MOR

```
public HashMap<String, Vector<Callback>> subscribers;  
public ROMImpl() throws RemoteException {  
    subscribers = new HashMap<String, Vector<Callback>>();  
}
```