

**POLYTECHNIQUE
MONTRÉAL**

LE GÉNIE
EN PREMIÈRE CLASSE



INF8480 - Systèmes répartis et infonuagique

TP1 : Appels de méthodes à distance

Présenté par :

Bineta Dieng 1678026

Papa Demba Tall 1712136

Présenté à :

Housseem Daoud

02 Octobre 2018

Introduction

L'objectif de ce laboratoire est de se familiariser avec les appels de procédure à distance à travers le Java Remote Method Invocation (RMI). Celui-ci permet de faire des appels de méthodes entre deux machines virtuelles Java qui peuvent s'exécuter sur deux hôtes différents. Dans la première partie, nous allons analyser le temps de réponse des appels RMI en fonction de la taille des arguments. La deuxième partie consistera en l'implémentation d'un système de fichiers partagés.

Partie 1: Comparaison de la performance des appels

Dans cette partie, nous avons démarré deux instances du serveur: une instance locale s'exécutant sur la même machine que le serveur ainsi qu'une instance distante.

server local:

```
[bidie@l4712-14 bin] $ rmiregistry &
[1] 30513
[bidie@l4712-14 bin] $ cd ..
[bidie@l4712-14 ResponseTime_Analyzer] $ ls
bin build.xml client.jar client.sh policy server.jar server.sh shared.jar src
[bidie@l4712-14 ResponseTime_Analyzer] $ ./server.sh
HELP:
./server.sh ip_address
- ip_address: (OPTIONAL) L'adresse ip du serveur.
  Si l'argument est non fourni, on considère que le serveur est local (ip_address = 127.0.0.1)
Server ready.
```

server distant:

```
ubuntu@inf4410-ubuntu-trusty-mini:~/ResponseTime_Analyzer$ ./server.sh 132.207.12.219
HELP:
./server.sh ip_address
- ip_address: (OPTIONAL) L'adresse ip du serveur.
  Si l'argument est non fourni, on considère que le serveur est local (ip_address = 127.0.0.1)
Server ready.
```

Client local:

```

[bidie@l4712-14 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 2677 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 884048 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 1099493 ns
Résultat appel RMI distant: 11
[bidie@l4712-14 ResponseTime_Analyzer] $ 

```

• Question 1 : Temps de réponse des appels RMI

Ci-joint le tableau récapitulatif des différents appels de méthodes en fonction de la variable x:

Valeurs de X	Appel normal (ns)	RMI local(ns)	RMI distant(ns)
1	3898	1401593	2136830
2	3949	1383470	2077039
3	4045	1063274	1890150
4	4491	1482072	2971271
5	9653	1198316	10852588
6	113505	2981523	88165227
7	2476705	14775137	880098321

Captures :

```
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 1
HELP:
./client.sh ip address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 3898 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 1401593 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 2136838 ns
Résultat appel RMI distant: 11
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 2
HELP:
./client.sh ip address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 3949 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 1383478 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 2877839 ns
Résultat appel RMI distant: 11
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 3
HELP:
./client.sh ip address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 4945 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 1863274 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 1890158 ns
Résultat appel RMI distant: 11
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 4
HELP:
./client.sh ip address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 4491 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 1482072 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 2971271 ns
Résultat appel RMI distant: 11
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 5
HELP:
./client.sh ip address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

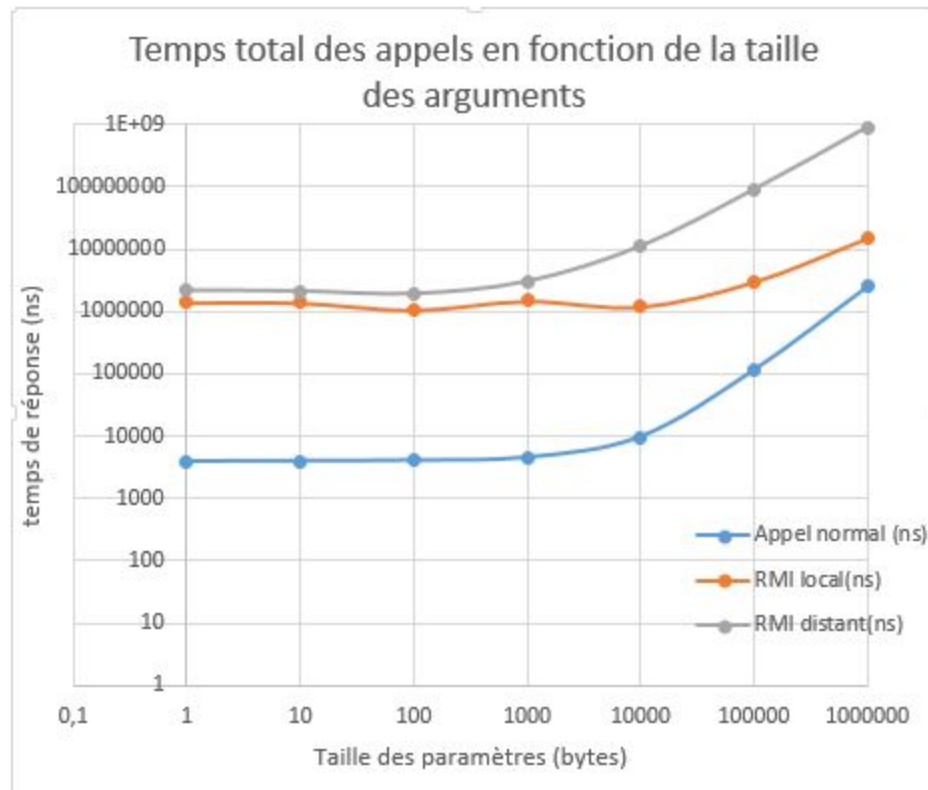
Temps écoulé appel normal: 9653 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 1198316 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 18852588 ns
Résultat appel RMI distant: 11
```

```
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 6
HELP:
./client.sh ip_address
- remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 113505 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 2981523 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 88165227 ns
Résultat appel RMI distant: 11
[patala@l4712-21 ResponseTime_Analyzer] $ ./client.sh 132.207.12.219 7
HELP:
./client.sh ip_address
- remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Temps écoulé appel normal: 2476705 ns
Résultat appel normal: 11
Temps écoulé appel RMI local: 14775137 ns
Résultat appel RMI local: 11
Temps écoulé appel RMI distant: 880098321 ns
Résultat appel RMI distant: 11
[patala@l4712-21 ResponseTime_Analyzer] $
```

Graphique de performance :



Commentaires:

Le comportement des différents appels RMI peut être décrit en 3 phases:

- **Pour des valeurs de X (puissance de 10 utilisée) entre 1 et 4** : l'appel normal est bien plus rapide que les appels RMI distant et local. Ces deux derniers ont des temps de réponse s'inscrivant dans le même ordre de grandeur.
- **Entre 4 et 5**: les temps de réponse des appels normaux et distant sont dix fois plus lent
- **Supérieures à 5**: tous les appels ralentissent. Le temps de réponse de l'appel RMI distant est plus important que celui des appels normal et RMI local.

Nous pouvons en déduire que l'appel RMI distant est donc moins performant lorsque la taille des paramètres est grande.

Avantages / inconvénients de JAVA RMI:

L'un des avantages de Java RMI est le fait qu'il soit orienté-objet. Ceci permet de passer des objets en paramètres comme des hashtables, et d'utiliser également les différents patrons de conception. Java RMI permet également de définir des serveurs facilement ainsi que les clients qui accèdent à ces serveurs, ce qui favorise la maintenabilité du code. Enfin, il intègre un garbage collector et permet également aux serveurs d'exploiter plusieurs threads Java.

Cependant, comme son nom l'indique, JAVA RMI ne peut pas être utilisé pour des systèmes autres que Java. De plus, nous avons constaté lors du TP que les appels RMI peuvent être moins performant lorsque la taille des paramètres est grande.

• Question 2: Description de l'interaction entre les composantes

❖ Client:

- Pour créer un objet de la classe client, la méthode *loadServerStub(string hostname)* permet de charger l'objet distant depuis le registre RMI à travers l'instruction : *stub = registry.lookup("server")* . "server" étant le nom utilisé côté serveur pour enregistrer l'objet dans le registre en question.
- Le client peut ainsi faire appel aux méthodes de la classe associée à l'objet (qui sont implémentées côté serveur).
- La méthode *run()* permet d'exécuter les méthodes.

❖ Serveur

- Une fois que le serveur est lancé à l'aide de la méthode *run()*, un objet est créé et exporté en interface distante (stub) grâce à la méthode *UnicastRemoteObject.exportObject(this)*.
- l'objet créé est enregistré dans le registre RMI et associé à un nom avec la méthode *registry.rebind("serveur", stub)*. Ce nom permettra au client de retrouver l'objet dans le registre RMI.

Partie 2: Système de fichiers à distance

Dans cette partie, nous avons implémenté un serveur de fichiers RMI permettant au client de créer un nouveau fichier(create), de récupérer le contenu d'un fichier (get), de consulter la liste des fichiers du serveur (list). Nous avons également mis en place un système d'authentification permettant de vérifier l'identité du client avant que celui-ci ne puisse exécuter des opérations.

Pour ce faire, nous avons créé une classe ProjectFile. Elle représente l'instance d'un fichier, avec toutes ses caractéristiques soient : le nom du fichier (Name), la somme de contrôle(checksum), l'identifiant du client (id), le contenu du fichier (content).

Scénario de fonctionnement :

- A chaque appel RMI, l'utilisateur doit fournir son nom d'utilisateur et son mot de passe
- le serveur authentifie l'utilisateur avec la méthode *verify(login, password)*
- A sa première connexion au serveur, l'utilisateur s'enregistre dans le service d'authentification avec la méthode *new(login, password)*

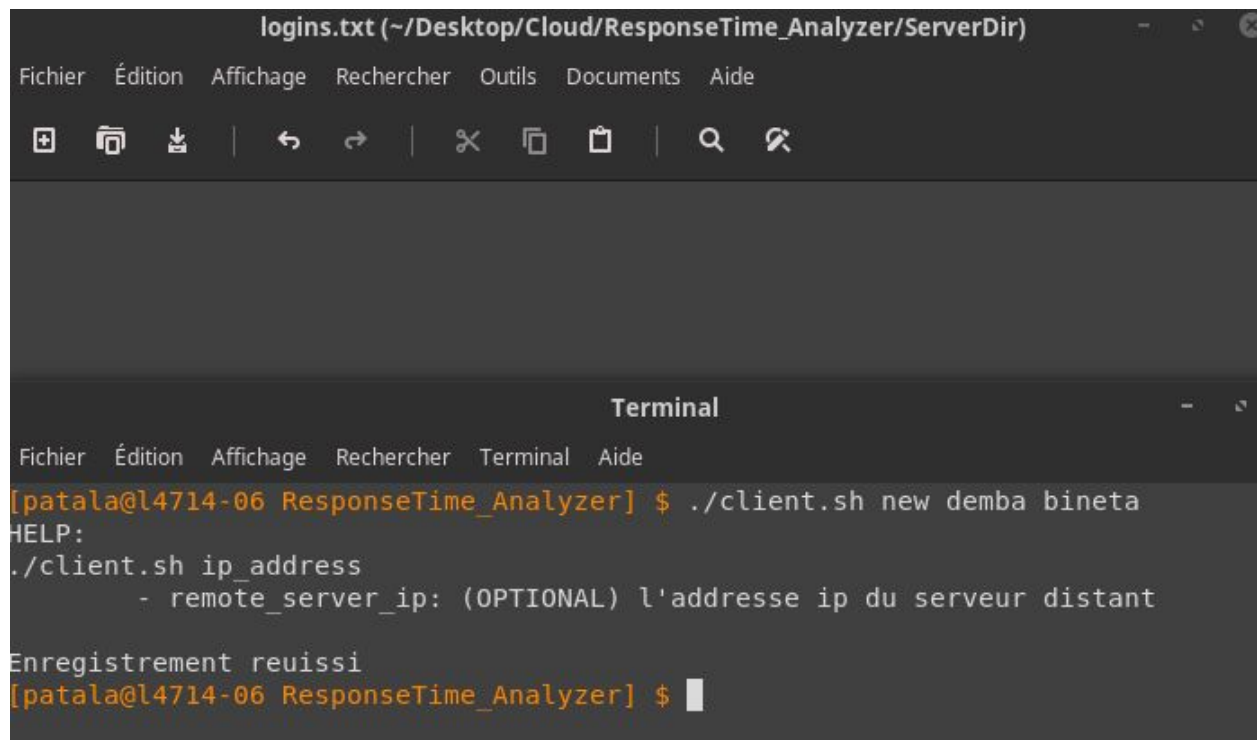
NB: Pour récupérer le contenu d'un fichier, le client appelle *get(nom, checksum)*. Le checksum permettra au serveur de vérifier si le fichier est déjà à jour ou non, pour éviter les transferts inutiles. Par ailleurs, le client doit aussi verrouiller le fichier pour pouvoir le modifier.

Test du service authentification

Avant ajout d'utilisateur, *verify* renvoie erreur et fichier vide

```
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh verify demba bineta
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

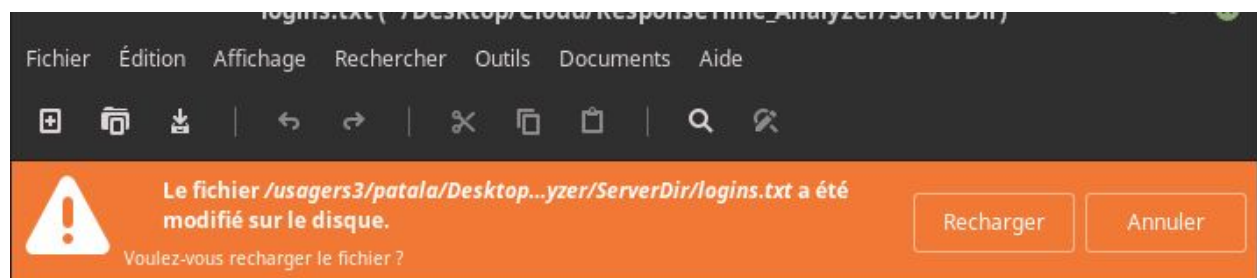
Cannot authenticate user
[patala@l4714-06 ResponseTime_Analyzer] $
```

The image shows a file editor window titled "logins.txt (~/Desktop/Cloud/ResponseTime_Analyzer/ServerDir)". The menu bar includes "Fichier", "Édition", "Affichage", "Rechercher", "Outils", "Documents", and "Aide". The toolbar contains icons for file operations. A terminal window is open within the editor, titled "Terminal", with its own menu bar: "Fichier", "Édition", "Affichage", "Rechercher", "Terminal", and "Aide". The terminal shows the following commands and output:

```
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh new demba bineta
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

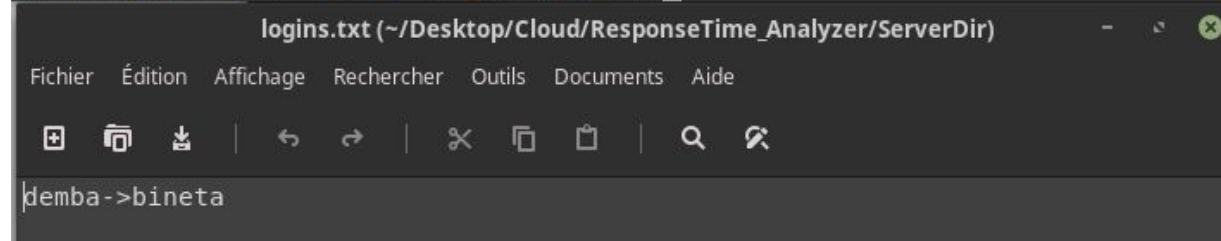
Enregistrement réussi
[patala@l4714-06 ResponseTime_Analyzer] $
```



Après ces commandes, il y a mise a jour du fichier et verify renvoie succès

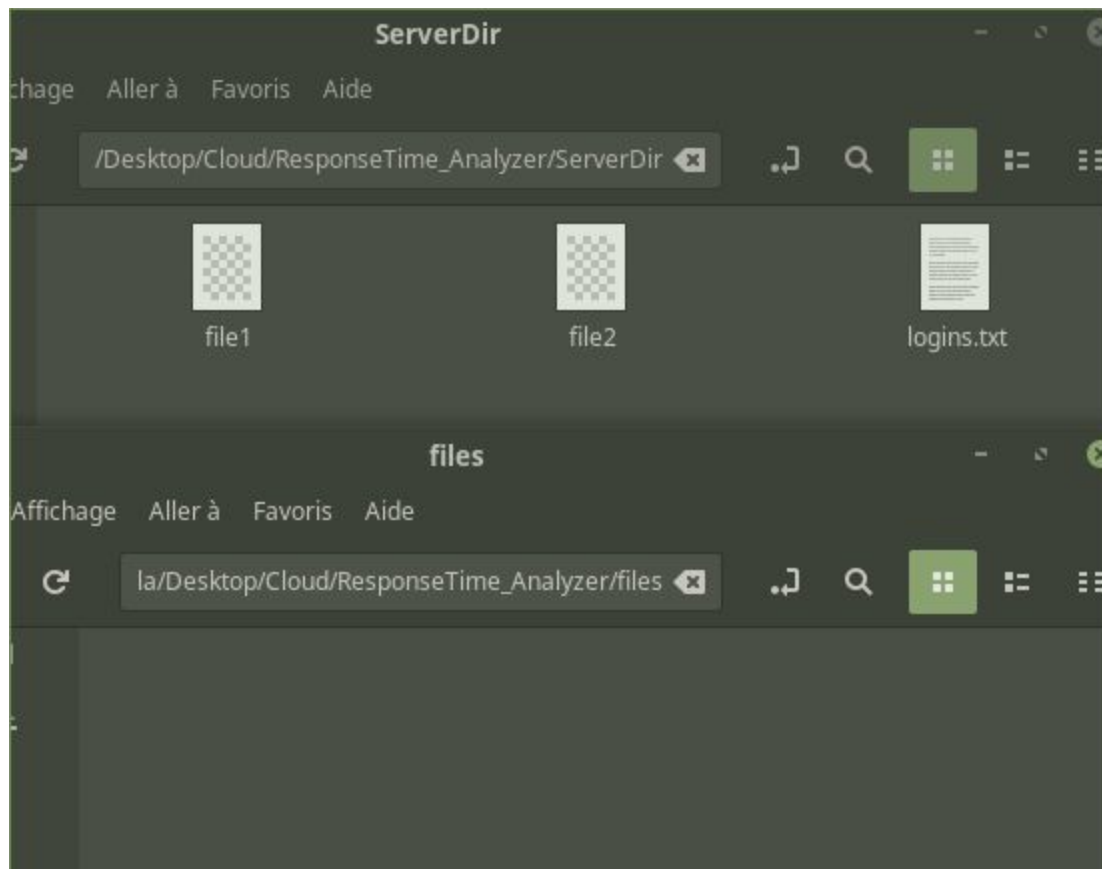

```
Enregistrement réussi
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh verify demba bineta
HELP:
./client.sh ip_address
  - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

The user is legit
[patala@l4714-06 ResponseTime_Analyzer] $
```



- **Scénario simple**

Etat des dossiers files (client) et serverDir apres appel a create et a list



```
The user is legit
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh create file1
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

file1 successfully created
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh create file1
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

file1 already exists. Choose different name
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh create file2
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

file2 successfully created
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh list
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

* file2 Non verouillee
* file1 Non verouillee
2 fichier(s)
```

Test des méthodes get, syncLocalDirectory, lock et push. Puis on s'assure que push ne fonctionne que si lock est exécutée (Exemple push file1)

```
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh get file1
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

file1 synchronise
[patala@l4714-06 ResponseTime_Analyzer] $ ls files/
file1
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh syncLocalDirectory
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Dossier local synchronise
[patala@l4714-06 ResponseTime_Analyzer] $ ls files/
file1 file2
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh list
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

* file2 Non verouillee
* file1 Non verouillee
2 fichier(s)
```

```

[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh lock file1
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Tentative de locker file1
Succes:file1 verrouillee par !886911
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh list
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

* file2 Non verrouillee
* file1 Verrouillee par l'utilisateur 886911
2 fichier(s)

[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh push file2
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Operation refusee : vous devez verrouiller d'abord le fichier
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh push file1
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Fichier envoye au serveur
[patala@l4714-06 ResponseTime_Analyzer] $ 

```

- Scénario avec conflit

Ceci est obtenu en dupliquant un autre dossier du projet et en lançant le 2e client dans un autre terminal.

(on mettra les captures par ordre d'exécution. Les noms des clients 1 et 2 sont inscrits sur les terminaux)

```
client 1
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh create monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

monfichier successfully created
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh list
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

* monfichier Non verouillee
1 fichier(s)

[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh lock monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Tentative de locker monfichier
Succes:monfichier verrouille par !208082
[patala@l4714-06 ResponseTime_Analyzer] $
```

```
client 2
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[patala@l4714-06 ResponseTime_Analyzer (copie)] $ ./client.sh create monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

monfichier already exists. Choose different name
[patala@l4714-06 ResponseTime_Analyzer (copie)] $ ./client.sh lock monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Tentative de locker monfichier
Erreur: Fichier deja verrouille par
[patala@l4714-06 ResponseTime_Analyzer (copie)] $
```



```
client 1
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh syncLocalDirectory
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Dossier local synchronise
[patala@l4714-06 ResponseTime_Analyzer] $ ./client.sh push monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Fichier envoye au serveur
[patala@l4714-06 ResponseTime_Analyzer] $
```

```
client 2
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[patala@l4714-06 ResponseTime_Analyzer (copie)] $ ./client.sh syncLocalDirectory
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Dossier local synchronise
[patala@l4714-06 ResponseTime_Analyzer (copie)] $ ./client.sh lock monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Tentative de locker monfichier
Succes:monfichier verrouille par !58526
[patala@l4714-06 ResponseTime_Analyzer (copie)] $ ./client.sh push monfichier
HELP:
./client.sh ip_address
    - remote_server_ip: (OPTIONAL) l'adresse ip du serveur distant

Fichier envoye au serveur
[patala@l4714-06 ResponseTime_Analyzer (copie)] $
```

Conclusion

Ce laboratoire nous a permis de nous familiariser avec les appels de procédure à distance à travers le Java Remote Method Invocation (RMI) dans le but de faire des appels de méthodes entre deux machines virtuelles Java qui peuvent s'exécuter sur deux hôtes différents.