

Compte Rendu SD_SE

-TUTO LANCEMENT

- naviguer dans le dossier source/Fournisseur
- ouvrir le code Fournisseur.java
- a la ligne 74, remplacer le dernier nombre par le numéro de la machine sur laquelle s'exécutera le fournisseur, (sur les exemples ci dessous, le numéro à remplacer est le 52)

```
73      try {  
74          System.setProperty("java.rmi.server.hostname","10.1.13.52");  
75          fournisseur = new Fournisseur();
```

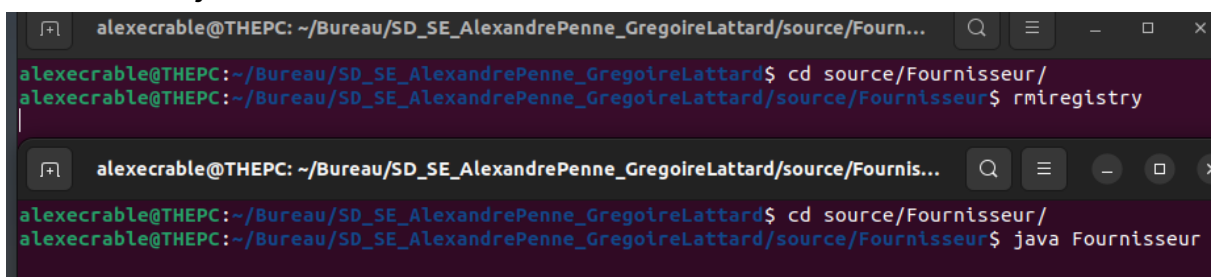
- naviguer dans le dossier source/Bar et faire la même manipulation à la ligne 48 de Commande.java

```
47      // Recuperation de la reference distante du serveur  
48      IBiere serveur = (IBiere) Naming.lookup("rmi://10.1.13.52/DedeLaChope");  
49
```

- revenir au dossier de base et lancer le makefile

I - LE FOURNISSEUR 💰💰💰

sur la machine du fournisseur, naviguez dans le dossier source/Fournisseur, lancez la commande `rmiregistry` puis lancez la commande `java Fournisseur` dans 2 terminaux différents comme ci :



```
alexecrable@THEPC: ~/Bureau/SD_SE_AlexandrePenne_GregoireLattard/source/Fourn...  
alexecrable@THEPC:~/Bureau/SD_SE_AlexandrePenne_GregoireLattard$ cd source/Fournisseur/  
alexecrable@THEPC:~/Bureau/SD_SE_AlexandrePenne_GregoireLattard/source/Fournisseur$ rmiregistry  
  
alexecrable@THEPC: ~/Bureau/SD_SE_AlexandrePenne_GregoireLattard/source/Fournis...  
alexecrable@THEPC:~/Bureau/SD_SE_AlexandrePenne_GregoireLattard$ cd source/Fournisseur/  
alexecrable@THEPC:~/Bureau/SD_SE_AlexandrePenne_GregoireLattard/source/Fournisseur$ java Fournisseur
```

II - LE BAR 🐱

sur la machine du barman, écrire `./bar.out [numéro de port]`

```
^C
alexecrable@THEPC:~/Bureau/SD_SE_AlexandrePenne_GregoireLattard$ ./bar.out 4577
```

III - LES CLIENTS 🍺🍺🍺

sur les machines clientes, écrire `./client.out [nom de machine barman] [numéro de port]`

```
alexecrable@THEPC:~/Bureau/SD_SE_AlexandrePenne_GregoireLattard$ ./client.out scinfe056 4577|
```

Répartition des tâches :

Nous nous sommes initialement répartis les tâches comme suit :

Alexandre :

- Client
- Main
- Communication

Gregoire :

- Controle
- Commande
- RMI

Cependant, nous avons passé tout le long du travail ensemble à se donner le relai sur le projet, il est donc difficile de déterminer une délimitation précise des tâches.

Description du travail :

DÉTAIL IMPORTANT :

Pour faire fonctionner le RMI, nous avons ajouté la ligne de code suivante dans le Fournisseur (trouvable dans le cours a la page 204 du chapitre 2) :

```
73      try {  
74          System.setProperty("java.rmi.server.hostname", "10.1.13.52");  
75          fournisseur = new Fournisseur();
```

I - Client (client.c Commande.c Demande.c Quitter.c)

Le client se connecte en tcp au processus de communication et peut envoyer 3 types de requêtes :

- une requête de demande d'informations demandant au barman la liste des bières dispo avec leur détail afin de l'afficher dans l'IHM du client (**Demande.c**)

- une requête de commande de bière lui permettant de demander une blonde ou une brune au barman, ainsi que la taille de la bière, il s'attend a plusieurs cas de réponses afin d'afficher différentes valeurs de résultat dans l'IHM du client (**Commande.c**)

- une requête d'adieu, permettant de sortir du bar..... :'((**Quitter.c**)

II - Communication

Le processus de communication possède un processus principal bouclant a l'infini sur une attente de connection de socket, lorsqu'il lit une nouvelle connection, il crée un processus fils à l'aide d'un fork sortant de cette boucle pour aller gérer le client correspondant, ce qui permet au barman d'accueillir plusieurs clients :)

Le processus fils créé attend ensuite que son client lui envoie une requête, puis en fonction de cette requête, il communiquera différentes informations au barman grâce aux pipes.

Pour ce faire, il crée un pipe avec pour nom son pid, puis envoie le nom au main à travers le pipe "transition" pour que le main puisse à son tour communiquer dans le pipe dédié

Il utilise ensuite ce pipe pour échanger des informations afin de répondre aux besoins du client

Si le client demande des infos, le processus de communication récupère les détails des 2 fûts du barman pour les envoyer au client

Si le client commande une bière, il demandera au barman de remplir une bière puis transmettra au client si la bière est bien servie ou pas.

Si le client quitte le bar, la communication ferme les sacs dédiés à ce client.

III - Main (Barman.c barcommande.c barinfo.c fifo.c biereverif.c sem.c)

Le main, s'occupe d'initialiser tout le Bar, il se charge de créer la mémoire partagée, les sémaphores et lance les programmes nécessaires au bar grâce à des fork :

- un pour lancer le processus de communication

- un fork qui restera dans une boucle infini permettant de donner la main entre le main, la communication et le contrôle.

après il rentre dans une boucle infini attendant que le processus de communication envoie une info dans le pipe de transition pour créer un processus fils utilisant les infos présent dans le pipe transition pour communiquer dans le pipe correspondant,

Si il doit communiquer des infos, il enverra les données des tireuses (**barinfo.c**)

Si il doit passer une bière (**barcommande.c**) il va passer le processus dans un ordonnanceur (**fifo.c**), cet ordonnanceur vérifie grâce à une mémoire partagée quel processus est le plus récent pour le faire passer, puis il va regarder si la quantité restante est suffisante par rapport à la quantité demandée et lancera une animation de remplissage

de verre, calculé pour faire pile poil les 4 ou 2 secondes d'attente de service (incroyable 🍺). **(biereverif.c)** la partie du remplissage de la bière est protégée par des sémaphores **(sem.c)**
En fonction de la réussite du remplissage, le processus renverra différentes valeurs au processus de communication pour savoir quel résultat transmettre au client.

IV - Controle (controle.c remplirfut.c)

Le processus de contrôle regarde si l'un des fûts est presque vide, puis, si l'un des fûts est effectivement vide, il va lancer le processus remplir fut avec un fork, pour que le processus puisse continuer de boucler afin de donner la main aux autres processus pour éviter que les clients ne se retrouvent bloqués.

cependant pour éviter de créer une infinité de processus fils gérant des processus commande, le Contrôle gère une mémoire partagée permettant de vérifier que la zone de contrôle est déjà en cours,

permettant donc de sauter le contrôle tout en continuant de boucler entre les 3 processus.

le processus remplir fut **(remplirfut.c)**, lance le processus commande à partir du terminal, puis créer des sockets udp pour se connecter au processus de Commande et se charge de choisir la bière à acheter une fois la bière achetée, remplirfut actualise les données de la tireuse concernée, en changeant le nom et le degré, et remet la quantité à 5L

V - Commande (Commande.java)

Le processus Commande, crée une socket serveur pour que remplirfut puisse se connecte à lui afin de lui dire quelle bière acheter, il récupère les données en rmi du fournisseur pour afficher la liste des bières disponibles pour permettre à remplir fut de choisir, une fois le choix fait, il invoque la méthode acheterbiere() et transmet les infos à remplirfut