

Documentation

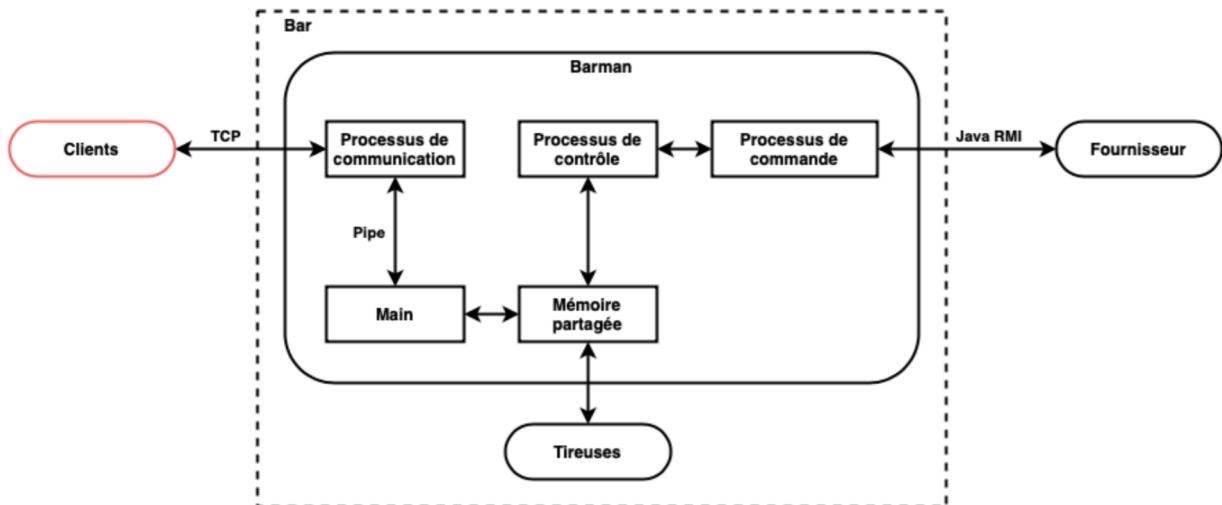
Projet de système distribué et système d'exploitation

UPPA PAU

27 avril 2023

Créé par : Sabrina LAVERGNE et Jonathan DERREZ

PARTIE CLIENT



Client.c

Ce fichier représente un client qui entre dans le bar. Il fait la connexion avec le bar, plus précisément avec le processus de communication, via une socket TCP.

Ce fichier se compile grâce à la commande : [make](#), et se lance avec la commande : [./commande.out adresse_port](#). Adresse et port sont ceux de l'ordinateur contenant le programme.

Pour cela, La fonction [creerSocketTCP\(\)](#) crée une socket TCP locale côté client, récupère l'adresse IP de la machine serveur, crée l'adresse de la socket destinataire et tente de se connecter au serveur en utilisant la fonction [connect\(\)](#). Cette fonction renvoie le descripteur de la socket ou -1 en cas d'erreur.

La fonction [recupererInformation\(\)](#) demande les informations au serveur en envoyant une requête de type INFORMATIONS. Elle crée une structure de requête, [struct requete](#), qui contient le type de la requête et la taille de la requête. La fonction crée également un message qui est une chaîne de caractères. La fonction envoie le message au serveur à l'aide de la fonction [write\(\)](#). Si l'envoi échoue, la fonction affiche un message d'erreur. Ensuite, la fonction attend une réponse du serveur à l'aide de la fonction [read\(\)](#). Si la lecture échoue, la fonction affiche un message d'erreur. Enfin, la fonction affiche les informations récupérées à l'aide de la structure tireuse.

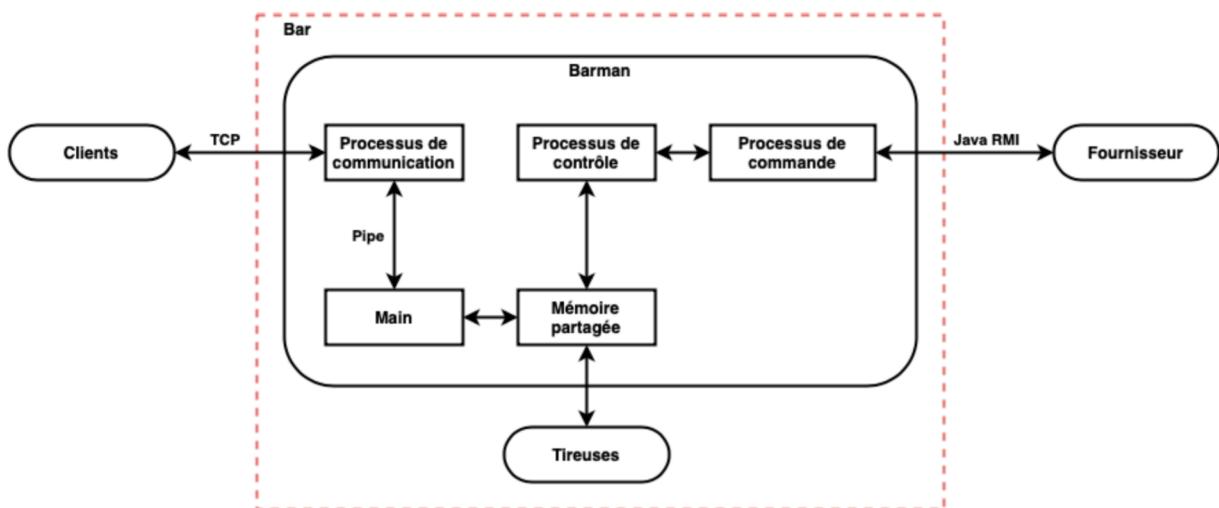
La fonction [demanderInformation\(\)](#) appelle la fonction [recupererInformation\(\)](#) et affiche les informations récupérées à l'écran sous la forme :

Les bières disponibles sont :
La bière : Kwak, Couleur : ambrée, Quantité restante : 5.0 L

La bière : Goudale, Couleur : blonde, Quantité restante : 5.0 L

La fonction `commanderBiere()` envoie une commande de bière au serveur en envoyant une requête de type **COMMANDÉ**. Elle crée une structure de requête, `struct requette`, qui contient le type de la requête et la taille de la requête. La fonction crée également une structure de tireuse qui représente la bière à commander. Elle envoie la requête et la structure de tireuse au serveur à l'aide de la fonction `write()`. Si l'envoi échoue, la fonction affiche un message d'erreur. Ensuite, la fonction attend une réponse du serveur à l'aide de la fonction `read()`. Si la lecture échoue, la fonction affiche un message d'erreur. Enfin, la fonction affiche la réponse du serveur à l'utilisateur.

PARTIE BAR



Le fichier `make` permet de compiler les fichiers `communication.c`, `main.c`, `contrôle.c`, `programme.c` et `tireuse.c` en même temps.

programme.c

Ce programme met en place trois threads : `main_thread`, `communication_thread` et `contrôle_thread`. Et un quatrième `scheduler_thread` qui bascule entre les trois autres. Ce fichier se lance avec la commande : `./programme.out adresse_ip port1 port2`. `Ip` et `port1` sont ceux du serveur `commande.java`. `port2` est celui ouvert pour le client.

La fonction `sigint_handler` est un gestionnaire de signal pour le signal `SIGINT`. Lorsque le signal est capturé (par exemple, lorsque l'utilisateur appuie sur Ctrl-C), `running` est mis à 0 pour signaler aux autres threads qu'il est temps de terminer. Les threads sont signalés avec `pthread_cond_signal`.

La méthode principale (main) crée les trois threads ([main thread](#), [communication thread](#), [contrôle thread](#)), puis le thread [scheduler thread](#) qui bascule entre les trois autres threads. Les trois threads sont ensuite joints avec [pthread join](#).

La fonction [switch threads](#) est exécutée par le thread [scheduler thread](#). Elle boucle indéfiniment jusqu'à ce que running soit mis à 0 par le gestionnaire de signal. À chaque itération, elle attend pendant une seconde (avec la fonction [sleep\(1\)](#)) puis elle envoie un signal à un thread différent, selon la valeur de [current thread](#). Elle met ensuite à jour [current thread](#) pour indiquer le prochain thread qui doit être exécuté.

communication.c

Ce fichier gère la communication entre le client qui entre dans le bar et le processus main. Ce fichier se lance automatiquement grâce au [communication thread](#) dans programme.c.

La méthode [communication thread\(\)](#) est responsable de la gestion de la communication entre le serveur et les clients. Elle crée une socket TCP sur le port 4000 et attend les connexions des clients. Une fois la connexion établie, elle crée un thread pour chaque client et, une fois terminé, le détache pour qu'il libère les ressources.

La méthode [communication thread handler\(\)](#) gère les signaux pour la fonction [communication thread\(\)](#). Elle ne fait rien, mais est nécessaire pour interrompre la fonction [pause\(\)](#).

La méthode [signal handler com\(\)](#) s'occupe des signaux pour l'ensemble du programme. Elle met la variable [running_com](#) à 0 pour arrêter le programme en cas de réception d'un signal.

La fonction [verificationCommande\(\)](#) vérifie la disponibilité de la bière demandée par le client. Elle prend en entrée les informations de la tireuse et de la commande et renvoie la réponse appropriée à [gererClientThread\(\)](#).

La fonction [verificationCommandeType\(\)](#) vérifie le type de bière demandé par le client. Elle prend en entrée la liste des tireuses et le nom de la bière et renvoie la tireuse correspondante.

La méthode [creerSocket\(\)](#) crée une socket sur le port spécifié et retourne le descripteur de fichier correspondant.

La méthode [gererClientThread\(\)](#) est responsable de la gestion des clients. Elle reçoit les requêtes des clients, vérifie la disponibilité de la bière et répond en conséquence.

Pour quitter l'exécution, il faudra faire quatre Ctrl+C pour arrêter les programmes [communication.c](#), [main.c](#) et [contrôle.c](#) en plus du [programme.c](#).

main.c

Ce fichier sert d'interface entre le processus de communication et la SHM (Shared Memory). Il est lancé automatiquement grâce au main thread dans programme.c

Nous avons une fonction vide main thread handler() qui sert à interrompre la méthode pause().

La fonction principale main thread() est une routine d'exécution qui est exécutée dans un thread séparé. Tout d'abord, elle appelle le programme ./Tireuse qui contient la structure struct tireuse.

Puis, elle initialise une clé, pour la sémaphore, à l'aide de la fonction ftok, crée un segment de mémoire partagée à l'aide de shmget, attache la zone de mémoire partagée à l'aide de shmat, crée un sémaphore à l'aide de create semaphore(), écrit dans un tube nommé, lit dans un autre tube nommé, et traite la commande enregistrée en dernier lieu. La fonction se termine lorsqu'un signal est capturé.

La fonction create semaphore() crée un ensemble de sémaphores, renvoie l'identifiant du sémaphore et initialise la valeur du sémaphore.

La fonction lock semaphore() verrouille le sémaphore spécifié.

La fonction unlock semaphore() déverrouille le sémaphore spécifié.

contrôle.c

Ce fichier permet de faire la connexion entre le fichier commande, en java, et la mémoire partagée. Il est lancé automatiquement grâce au contrôle thread dans programme.c

Il y a d'abord la création de la socket UDP qui est reliée au programme commande.Java, puis la connexion avec la mémoire partagée SHM. Ce fichier récupère la structure tireuse, puis créé la sémaphore grâce à la méthode create semaphore(). Il verrouille la sémaphore grâce à la clé le temps de faire ses instructions puis déverrouille la sémaphore à la fin du code.

Pendant que le thread contrôle tourne (lancé par programme.c), il vérifie l'état des deux tireuses. Si la tireuse[0] est vide, il envoie au processus commande.java, le message "ACHETERAMBREE", si c'est la tireuse[1], il envoie le message "ACHETERBLONDE" grâce à la commande sendto().

Une fois le message envoyé, nous attendons la réponse du processus commande.java avec la méthode recvfrom(), ce message sera sous la forme :

Bières ambrées :

- [1] Kwak
- [2] Mousse Ta Shuc
- [3] Queue de Charrue

OU

Bières blondes :

- [1] Paix Dieu
- [2] Goudale
- [3] Delirium Tremens

Quelle bière voulez-vous acheter ?

Quelle bière voulez-vous acheter ?

L'utilisateur du processus contrôle.c choisi sa bière avec 1, 2 ou 3 puis le message est à nouveau envoyé au processus commande.java.



Une fois la validation reçus (le fût de bière), le programme met à jour la SHM en récupérant le type de la bière et en modifiant le nom et la quantité de la tireuse correspondante.

Puis le programme attend pendant deux secondes grâce à la méthode [sleep\(2\)](#) avant de vérifier l'état des fûts.

A la fin du programme, nous déverrouillons la sémaphore, libérons la mémoire de la variable bière puis fermons la socket.

tireuse.c

Il s'agit d'un programme qui décrit le fonctionnement d'une tireuse de bière et qui utilise des sémaphores et des segments de mémoire partagée pour la synchronisation et la communication entre les différents processus.

Le programme commence par inclure les bibliothèques nécessaires pour les appels système et le fichier d'en-tête [Tireuse.h](#) qui définit la structure tireuse.

Ensuite, le programme définit un [union semun](#) pour définir les arguments passés à [semctl](#), qui permet de contrôler les sémaphores, puis il définit plusieurs fonctions pour créer, verrouiller et déverrouiller des sémaphores, ainsi que pour supprimer un sémaphore. Les fonctions utilisent les appels système [semget](#), [semctl](#) et [semop](#).

La fonction [creer_shm\(\)](#) crée un segment de mémoire partagée avec la clé 5, initialise les données d'une structure tireuse et les stocke dans le segment de mémoire partagée. Elle utilise les appels système [shmget](#), [shmat](#), [strcpy](#) et [shmdt](#).

La fonction [main\(\)](#) utilise la fonction [erase_semaphore](#) pour supprimer un sémaphore avec la même clé que le segment de mémoire partagée. Elle utilise ensuite la fonction [shmget](#) pour créer un segment de mémoire partagée de taille [2*sizeof\(struct tireuse\)](#) avec la même clé et les permissions [0666 | IPC_CREAT](#). Elle utilise la fonction [shmat](#) pour attacher le segment de mémoire partagée à la mémoire du processus et elle stocke le pointeur renvoyé dans une variable tireuse de type struct tireuse. Elle utilise la fonction [create_semaphore](#) pour créer un sémaphore avec la même clé et elle stocke l'ID du sémaphore renvoyé dans une variable [semid](#).

Enfin, elle utilise l'exemple d'utilisation de la fonction [lock_semaphore](#) pour verrouiller le sémaphore, modifie les données de la première structure tireuse dans le segment de mémoire partagée en utilisant la fonction [strcpy](#) et [tireuse\[0\].quantite_biere](#), puis déverrouille le sémaphore en utilisant la fonction [unlock_semaphore](#).

Commande.java

Ce fichier fait le lien entre le processus `contrôle.c`, via une socket UDP, et le fichier `Fournisseur.java`, via du Java RMI. Ce fichier se compile grâce à la commande : `make`, et se lance avec la commande : `java Commande adresse port`. L'adresse du serveur Fournisseur et le port ouvert pour le barman.

Tout d'abord, le programme lance la connexion avec la socket UDP. Puis, il récupère le message du processus `contrôle.c` qui lui envoi "ACHETERAMBREE" ou "ACHETERBLONDE".

Puis, le fichier fait la connexion en Java RMI avec `Fournisseur.java` grâce à l'objet distant "`DedeLaChope`", pour pouvoir récupérer les bières.

Selon le message reçu, le fichier renvoie le message :

Bières ambrées :

- [1] Kwak
- [2] Mousse Ta Shuc
- [3] Queue de Charrue

OU

Bières blondes :

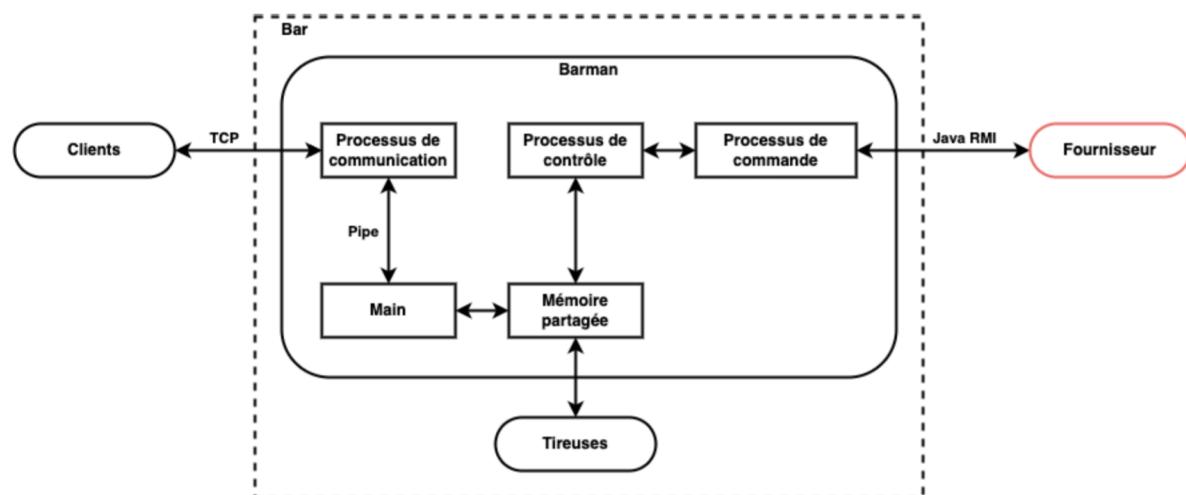
- [1] Paix Dieu
- [2] Goudale
- [3] Delirium Tremens

Quelle bière voulez-vous acheter ?

Quelle bière voulez-vous acheter ?

Puis attend la réponse de `contrôle.c` et renvoie la validation (le fût de bière).

PARTIE FOURNISSEUR



Ce fichier se compile grâce à la commande : `make` puis `rmiregistry&`, et se lance avec la commande : `java Fournisseur adresse`. L'adresse est celle de l'ordinateur sur lequel est le programme.

Nous avons rajouté une ligne dans le programme `Fournisseur.java` :

```
System.setProperty( "java.rmi.server.hostname" , argv[0] ); // Ajouté
```

Qui nous sert à récupérer l'adresse du serveur.

SCENARIO DE TEST

Un client arrive et demande la liste des ambrées

Tireuse[0] : Kwak, 2L

Un client arrive et demande la liste des ambrées

Tireuse[1] : Goudale, 0,5L

Il décide de prendre une pinte de Kwak.

Tireuse[0] : Kwak, 1.5L

Un nouveau client arrive et demande un demi de Goudale.

Tireuse[1] : Goudale, 0,25L

Puis il redemande une pinte de Goudale, le barman le prévient qu'il n'y a plus assez de littres.

Il finit par choisir un demi de Goudale.

Tireuse[1] : Goudale, 0L

Le fut est vide, on passe commande, le barman choisi la blonde qu'il veut, il choisit la Paix Dieu, le fut se rempli de 5L.

Tireuse[1] : Paix Dieu, 5L

Le premier client redemande ce qu'il y a comme blondes

Puis il commande une pinte de Paix Dieu.

Tireuse[1] : Paix Dieu, 4.5sL