Projet de programmation réseau

José Vander Meulen, Anthony Legrand, Bernard Frank, Iyad Khaddam

Ce projet consiste à construire un système distribué simulant le fonctionnement d'une banque. Il permettra de simuler un nombre d'opérations relativement limitées qui sont généralement effectuées par les clients d'une banque. Comme ce système est un simulateur, il ne gérera pas les aspects de sécurité inhérents à un système bancaire réel et ne fonctionnera pas toujours de manière réaliste. Il sera composé de différents programmes qui ne seront potentiellement pas tous exécutés sur une même machine.

C'est un projet à réaliser par groupe de deux étudiants durant les semaines 10, 11 et 12 de ce quadrimestre. Nous vous laissons le choix de former les groupes. Néanmoins, vous devez choisir un partenaire qui assiste chaque semaine aux deux mêmes séances que vous. Si une séance contient un nombre impair d'étudiants, nous accepterons un groupe de trois étudiants par séance. Notez que votre présence en séance est requise durant les semaines 10, 11 et 12.

Description du projet.

Le système à développer est un programme distribué qui simule certaines opérations bancaires. Il est composé d'un serveur central et d'applications clientes qui se connecteront au serveur central pour effectuer leurs opérations. Dans la suite de ce document, nous avons tenté de délimiter le plus précisément possible les différents composants du simulateur, afin de vous laisser vous focaliser sur les aspects techniques du système et de vous permettre de le construire dans le temps imparti.

Le système est composé d'un serveur central qui représentera la banque et qui contiendra un livre de compte maintenant le solde des clients. Parallèlement à ce serveur central, des applications clientes se connecteront à distance pour effectuer des virements d'un compte à l'autre. La figure 1 présente une vue schématique du simulateur.

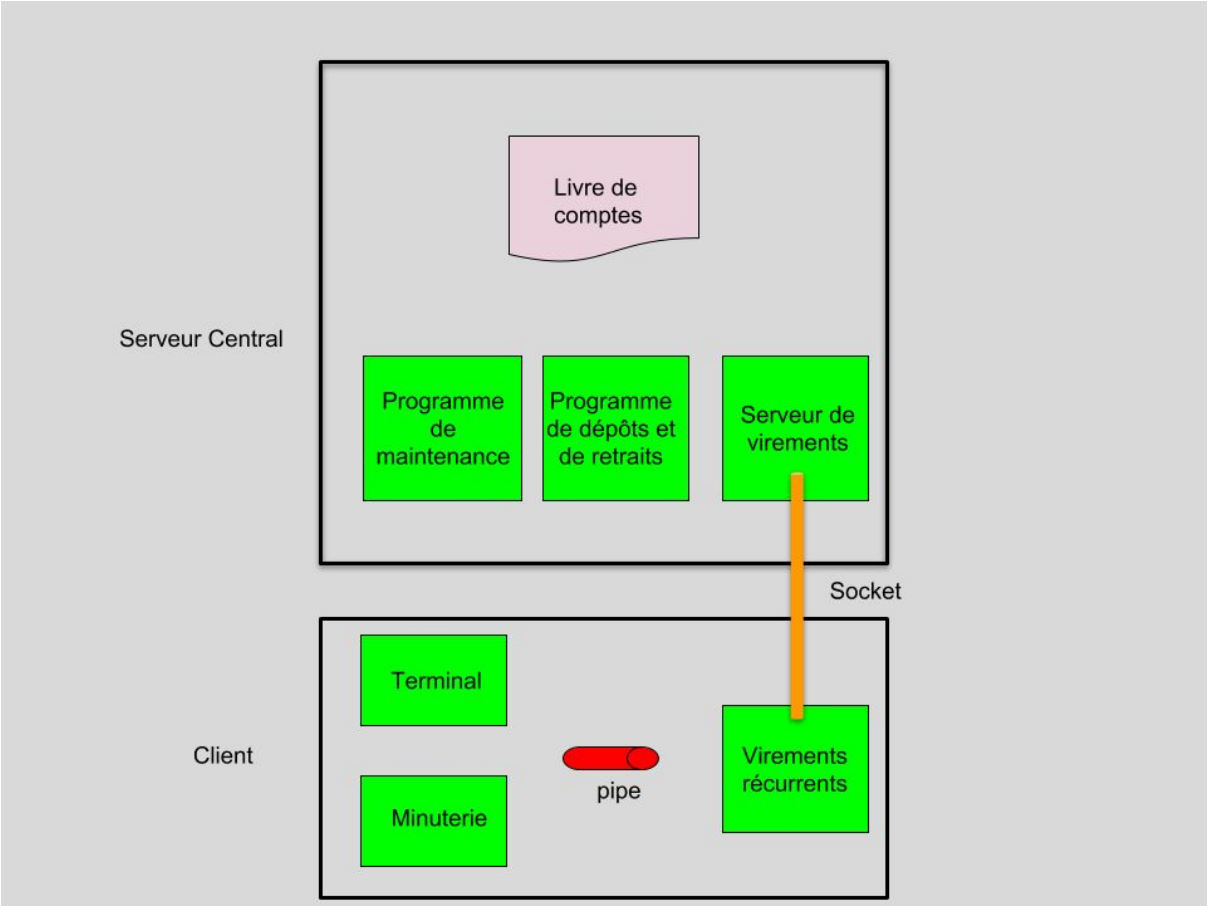

Figure 1

Le serveur central

Le serveur central permet de simuler la banque. En pratique, il gère un livre de comptes contenant le solde d'exactement 1000 comptes bancaires portant les **numéros** 0, 1, 2, …, 999. Ce livre de comptes est représenté de manière informatique à l'aide d'une **mémoire partagée**. Chaque **solde** est représenté par un entier qui peut être positif ou négatif si le titulaire du compte est à découvert.

Le livre de compte étant une mémoire partagée, vous devrez veiller à **protéger les accès concurrents** à cette mémoire à l'aide de sémaphores.

Le serveur central offre **3 programmes** qui permettent de gérer les comptes bancaires:

1. Un **programme de gestion des dépôts et des retraits** bancaires.
2. Un **serveur de virements** qui offre la possibilité d'effectuer des virements entre deux comptes.
3. Un **programme de maintenance** qui permet de simuler des opérations de maintenance sur le livre de compte.

Le programme de dépôts et de retraits.

Le programme de gestion des dépôts et de retraits est un programme qui offre un moyen très simple d'effectuer un dépôt ou un retrait sur un compte donné. En pratique, c'est un programme dont le nom est "`pdr`" qui prend deux arguments: un numéro de compte et un montant.

1. Si le montant est positif, le programme effectue un dépôt sur le compte en banque et affiche le nouveau solde du compte. Par exemple, l'exécution de la commande "`pdr 24 200`" permet d'effectuer un dépôt de 200 euros sur le compte 24.
2. Si le montant est négatif, le programme effectue un retrait sur le compte en banque et affiche le nouveau solde du compte. L'exécution de la commande "`pdr 24 -200`" permet d'effectuer un retrait de 200 euros sur le compte 24.

Pour limiter le temps de développement et aller à l'essentiel, vous pouvez supposer que les deux arguments représentent des entiers valides. Si ce n'est pas le cas, le comportement de votre programme est indéterminé. Pour les mêmes raisons, ce programme n'offre volontairement que des fonctionnalités basiques. Néanmoins, il n'est pas trivial dans le sens où il doit accéder de manière concurrente à une mémoire partagée. Pour ce faire, il doit protéger les accès au livre de compte partagé à l'aide de sémaphores.

Le serveur de virements

Le serveur de virements permet d'effectuer des virements bancaires entre deux comptes. C'est un serveur dans le sens où il n'offre pas de moyen direct de faire des virements mais il faut obligatoirement passer par un client informatique pour pouvoir effectuer un virement. Un client informatique doit effectuer les opérations suivantes lorsqu'il souhaite émettre un ou plusieurs virements:

1. Ouvrir une connection TCP.
2. Envoyer les informations concernant le ou les virements bancaires à effectuer. Pour chaque virement, le serveur a besoin d'un numéro de compte émetteur, d'un numéro de compte bénéficiaire et d'un montant à transférer.
3. Fermer la connection TCP.

En pratique, c'est un programme dont le nom est "`server`" qui prend comme argument uniquement le port sur lequel se connecte le serveur. Vous pouvez supposer que l'argument qui représente le port est un entier naturel valide. Si ce n'est pas le cas, le comportement de votre programme est indéterminé.

Comme pour le programme de dépôts et de retraits, pour limiter le temps de développement, ce programme n'offre volontairement que des fonctionnalités basiques. Néanmoins, il n'est pas non plus trivial car il manipule des "sockets" et il doit également accéder de manière concurrente à la mémoire partagée relative au livre de compte.

Le programme de maintenance

Le programme de maintenance permet de simuler des opérations de maintenance sur le livre de compte.

Concrètement, c'est un programme dont l'appel prend la forme "`maint type [opt]`". Il prend en argument un entier représentant le type de l'opération à effectuer et éventuellement un argument supplémentaire qui sera nécessaire pour une des opérations de maintenance:

1. Si "`type`" est égal à 1, il **crée les ressources partagées** relatives au livre de comptes tels que la mémoire partagée et les sémaphores.
2. Si "`type`" est égal à 2, il détruit **les ressources partagées** relatives au livre de comptes.
3. Si "`type`" est égal à 3, il **réserve de façon exclusive le livre de comptes partagé** pour une période de temps donné. La durée de cette période est fournie au programme de maintenance par le paramètre "`opt`" qui représente le nombre de secondes durant lequel le programme réserve de façon exclusive le livre de comptes. Notez que pour vous simplifier la vie, vous pouvez supposer que "`opt`" représente un entier naturel représentable en C. Si ce n'est pas le cas, le comportement de votre programme est indéterminé. Pour construire cette fonctionnalité, pensez à utiliser la fonction "`sleep`". L'intérêt essentiel de cette option est qu'elle offre un moyen effectif de tester les accès concurrents au livre de compte. En effet, lorsque le programme de maintenance accède au livre de comptes de manière exclusive, ni le serveur de virements, ni le programme de dépôts et de retraits ne peuvent accéder à celui-ci.

Les clients

Afin d'effectuer des virements, le système comprend également des clients informatiques. Ceux-ci sont composés d'un programme "père" qui met à disposition de l'utilisateur final un "prompt" qui offre des commandes permettant de gérer des virements, d'un "fils minuterie" qui génère un battement de coeur à intervalle régulier et d'un programme "fils" qui gère une série de virements récurrents émis à chaque battement de coeur.

Les systèmes bancaires réels offrent généralement la possibilité d'effectuer des virements récurrents qui sont couramment des ordres permanents. En pratique ces ordres permanents sont gérés par un des serveurs bancaires plutôt que par un client. Au contraire, dans votre simulateur, c'est le client qui doit gérer les virements récurrents.

Concrètement, un client est un programme dont le nom est "`client`" qui prend quatre arguments: "`adr`", "`port`", "`num`" et "`delay`" (où `port`, `num` et `delay` sont des entiers naturels). Les arguments "`adr`" et "`port`" représentent respectivement l'adresse et le port du serveur central. L'argument "`num`" représente le numéro du compte émetteur des virements effectués par le client. Tous les virements de ce client auront donc le même

émetteur. L'argument "`delay`" représente l'intervalle de temps (en secondes) entre deux émissions des virements récurrents.

Le père et les fils communiquent ensemble à l'aide d'un "pipe". Le père et le fils communiquent avec le serveur de virements à l'aide de connections TCP.

Le père présente à l'utilisateur final un prompt permettant d'exécuter trois types de commandes:

1. une commande "`+ n2 somme`" (où `n2` et `somme` sont des entiers naturels) qui permet d'effectuer un virement entre le compte "`num`" du client et le compte "`n2`" d'un montant équivalent à somme.
2. une commande "`* n2 somme`" (où `n2` et `somme` sont des entiers naturels) qui transmet au fils un nouveau virement entre le compte "`num`" du client et le compte "`n2`" d'un montant équivalent à somme. Ce virement récurrent est émis toutes les "`delay`" secondes par le fils.
3. une commande "`q`" qui déconnecte le client et libère les ressources du clients.

Notez que techniquement, nous vous imposons que, pour chaque lot de virements, le client établit une nouvelle connection TCP avec le serveur.

Comme pour les programmes liés au serveur, les clients n'offrent volontairement que des fonctionnalités basiques. Néanmoins, ils ne sont pas non plus triviaux car ils sont constitués de plusieurs processus communiquant à l'aide de "pipes". Ils utilisent également des "sockets".

Gestion des arrêts et des situations exceptionnelles du serveur et des clients

De manière générale, ni le serveur, ni le client ne doivent gérer les arrêts brutaux. En particulier, nous supposerons que personne n'effectuera un "`kill -9`" pour "tuer" un de vos programmes. Vous ne devez donc pas gérer ces situations d'arrêts brutaux.

Par contre, il est prévu que votre serveur de virements soit arrêté lorsque l'utilisateur appuie sur les touches "ctrl + c". Lorsque c'est le cas, si votre serveur traite un lot de virements, il termine son traitement courant et ensuite s'arrête, sinon il s'arrête immédiatement.

Vous ne devez pas gérer de manière subtile les situations exceptionnelles (eg. impossibilité de créer un socket, erreur d'écriture sur un pipe ou un socket, …). Lorsqu'un tel cas arrive, vos programmes affichent un message d'erreur avec la fonction "`perror`" et se termine de manière abrupte sans se soucier de libérer les ressources.

Lors d'arrêts brutaux d'un programme, il est possible que certains "ipcs" non-utilisés soient encore réservés sur la machine sur laquelle tournait le programme. Pensez à utiliser les commandes "ipcs" et "ipcrm" pour traiter ces cas.

Délivrables et tests de votre programme

Pour le début de la deuxième séance de la semaine 10, nous vous demandons de produire et de nous remettre en main propre un document, de 2 faces A4, décrivant l'architecture de votre programme et la manière dont vous allez organiser la découpe en modules. En pratique, la production de ce document pourra être perçue comme la phase d'analyse de votre programme.

Pour le samedi 12/05/2018 à 23h59, nous vous demandons de soumettre sur Moovin tous les fichiers sources relatifs à votre application, ainsi qu'un *makefile*. Pour information, nous utiliserons un logiciel anti-plagiat pour nous aider à détecter les éventuels plagiats entre les groupes.

Durant le courant de la semaine 13, nous organiserons une séance durant laquelle nous testerons vos simulateurs. Notez que nous testerons exclusivement l'application que vous avez soumise sur Moovin. En plus de ces tests, nous organiserons, à la suite des tests, une rencontre avec chaque groupe pour discuter de votre implémentation.