BASTIN Max
SURGET Jérémy

23/04/2019

Simulation d'un système de paiement par carte bancaire

ESIEE PARIS

M. COUSTY

SOMMAIRE

I. Introduction 3

II. Découpage de la semaine 4-10

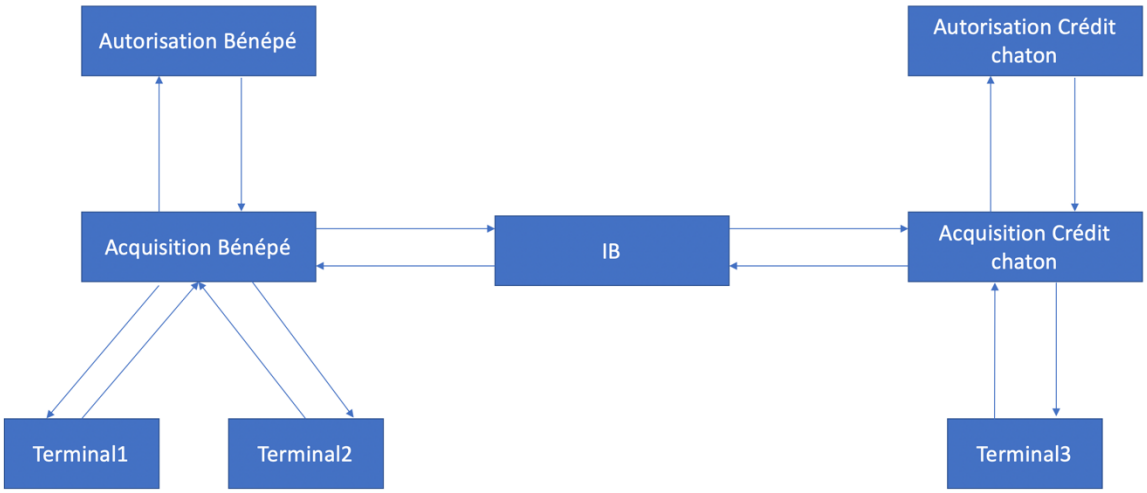
III. Limitations de notre simulation et bugs rencontrés 11-12

IV. Annexe : Réponse aux questions 13-14

I. Introduction

Lien du GitHub : https://github.com/jeremySrgt/reseauBancaire

Pour ce projet le but était de simuler un système de paiement par carte bancaire qui marcherait de la façon suivante. Un utilisateur fait une demande de paiement à partir d'un terminal et il reçoit l'autorisation ou non d'effectuer son paiement. En technique cela consiste à créer plusieurs processus qui communiquent entre eux grâce à des tubes et des threads permettant d'assurer le parallélisme. On pourra schématiser ceci de la façon suivante :



Les flèches représentent des tubes, les threads ne sont pas représentés ici

Nous avons travaillé à deux sur ce projet et afin de réaliser ce projet dans les meilleures conditions nous avons utilisé l'outil GitHub qui permet un versionning du code, de participer à plusieurs sur un projet, de récupérer les sauvegardes et les dernières nouveautés ajouté par les membres du projet. De plus il permet d'avoir une traçabilité du code et de à tout moment du projet pouvoir revenir à une version fonctionnelle.
Nous faisions également un point tous les matins avec M. Cousty sur les tâches que nous avions accomplies le jour précédent et ce qu'il faudrait faire pour le lendemain, nous pouvions ensuite nous répartir les tâches afin d'être le plus efficace possible.

II. Découpage de la semaine

Jour 1 :

Durant ce premier jour nous avons commencé par une lecture approfondit du sujet afin de nous l'approprier le plus possible. Suite à cela nous avons répondu à quelques questions afin d'améliorer notre connaissance et vision globale du sujet, la réponse à ces questions sont disponibles en annexe.

Ensuite nous avons commencé par coder le terminal. Cette première version du terminal avait juste pour but de rediriger les sorties standards, afin de prendre une demande de carte bancaire que nous écrivions dans le terminal directement. Puis nous avons créé un fichier de texte contenant juste au début des numéros de carte de crédit (16 chiffres) et avec la fonction litLigne() qui nous était fourni nous lisions une ligne au hasard pour prendre un numéro de CB aléatoire à chaque fois. Suite à cela nous avons créé acquisition qui pour l'instant récupère juste les envois du terminal. Pour que les deux processus puissent s'échanger des messages nous utilisons des tubes, un pour écrire et un pour lire.

Jour 2 :

Le but était ici d'ajouter le serveur autorisation qui permettrait d'autoriser ou non les paiements.Il fallait donc réussir à transmettre les demandes du terminal à travers le serveur acquisition qui lui transmettait ensuite la demande au serveur autorisation. Le serveur acquisition est le processus qui crée le terminal et le serveur autorisation, il fallait donc améliorer notre version d'acquisition.

Modification acquisition : Nous avons dû ajouter des tubes pour relier acquisition et autorisation. Nous avons procédé de façon similaire à celle utiliser pour relier acquisition et terminal. Désormais acquisition utilise des forks afin de créer les 2 processus acquisition et terminal. Notre programme acquisition prend en paramètre le nombre de terminaux à créer, pour ensuite via une boucle while fork le nombre de terminaux que l'on souhaite.

Ajout Autorisation : dans la première version d'autorisation il n'y a pas encore de tube, on redirige les sorties standards pour tester le bon fonctionnement. Au début le message qui devra venir du serveur acquisition sera rentré par nous à la main, il faudra rentrer un message selon la bonne forme (|CodeCB|Message|Montant|). Nous utilisons la fonction découpe afin de découper le message reçu pour traiter ses informations, cela nous permet de séparer dans des variables le code de la carte, la demande et le montant. Dans cette version nous avons codé en dure un numéro de Carte dans une variable, nous allons comparer le numéro reçu (par la fonction découpe) à ce numéro. S'ils sont identiques nous afficherons un message puis vérifierons que la demande de retrait est inférieure au solde du compte (codé en dure également). Une fois cela fait nous affichons un message formaté grâce à la fonction fournis message(), avec le numéro de carte, la réponse et la validation ou non du retrait (1 ou 0).
Maintenant que cela fonctionne nous essayons d'ajouter les tubes et de relier le serveur acquisition au serveur autorisation.

Modification terminal : Dans la première version la comparaison sur le message retour d'acquisition se faisait uniquement sur une valeur (1 ou 0) et affichait le message correspondant, nous avons améliorer cela et désormais la comparaison est faite selon la découpe du message demandé dans le sujet (XXXXXXXXX|XXXX|X|).

A la fin de ce jour nous rencontrons un problème, lors de l'échange entre les processus nous faisons beaucoup d'affichage afin de regarder que tous se passe bien et en cas de bug trouver plus vite d'où vient le problème, ici le message que reçoit acquisition venant d'autorisation a été dégradé et la comparaison pour le terminal n'est plus possible.

```
nous sommes dans le processus 'acquisition'
processus autorisation cree par acquisition
terminal cree par acquisition
terminal cree par acquisition
on est dans autorisationjuste avant ecritligne de terminal
juste avant litligne de terminal
juste avant ecritligne de terminal
juste avant litligne de terminal
la demande envoyé PAR le terminal est : |1234123412341234|Demande|17|
message envoyé par autorisation a acquisition : |1234123412341234|Reponse|1|
reponse autorisation : |1234123412341234|Reponse|1|
reponse recu de acquisition apres traitement par autorisation : 24123412341345
------------------------------payement refuse-----------------------------------
```

Comme nous pouvons le voir sur l'avant dernière ligne, la réponse reçue ne ressemble plus du tout au formatage attendu. Cela était dû à un oubli de saut de ligne dans un printf et donc la ligne en question était écrite dans le tube car elle n'était pas flush automatiquement par le saut de ligne.

Jour 3 :

Nous avons réussi à résoudre le problème que nous avions rencontré lors de la journée précédente. Une fois ce problème résolu nous avons améliorer les noms de variable pour plus de clarté et changé le message lors d'un refus de paiement.

```
nous sommes dans le processus 'acquisition'
terminal cree par acquisition
processus autorisation cree par acquisition
on est dans autorisation
juste avant ecritligne de terminal
juste avant litligne de terminal
la demande envoyé PAR le terminal est : |1234123412341234|Demande|17|
message envoyé par autorisation a acquisition : |1234123412341234|Reponse|1|
reponse de autorisation : |1234123412341234|Reponse|1|
reponse recu de acquisition apres traitement par autorisation : |1234123412341234|Reponse|1|
------------------------payement accepte--------------------------
```

Jour 4 :

Nous avons introduit l'utilisation et la création d'un annuaire et la possibilité de lancer plusieurs terminaux de paiement en même temps.

Possibilité plusieurs terminaux : Pour pouvoir lancer plusieurs terminaux nous laissons l'utilisateur saisir le nombre de terminaux qu'il veut lancer dans la simulation. Ensuite dans le code il s'agit de faire une boucle while avec un fork à l'intérieur pour créer le nombre de terminaux désiré. Grâce à deux tableaux de tubes nous assignons à chaque terminal créer un tube en lecture et un tube en écriture que nous lui passons via le execlp pour lancer son processus.
Par la suite nous avons ajouté des threads, chaque terminal se voit relier à un thread pour lui permettre de communiquer avec acquisition. Pour ce faire nous avons utilisé un tableau de threads. Nous détaillons plus loin dans le rapport pourquoi avoir ajouté des threads.

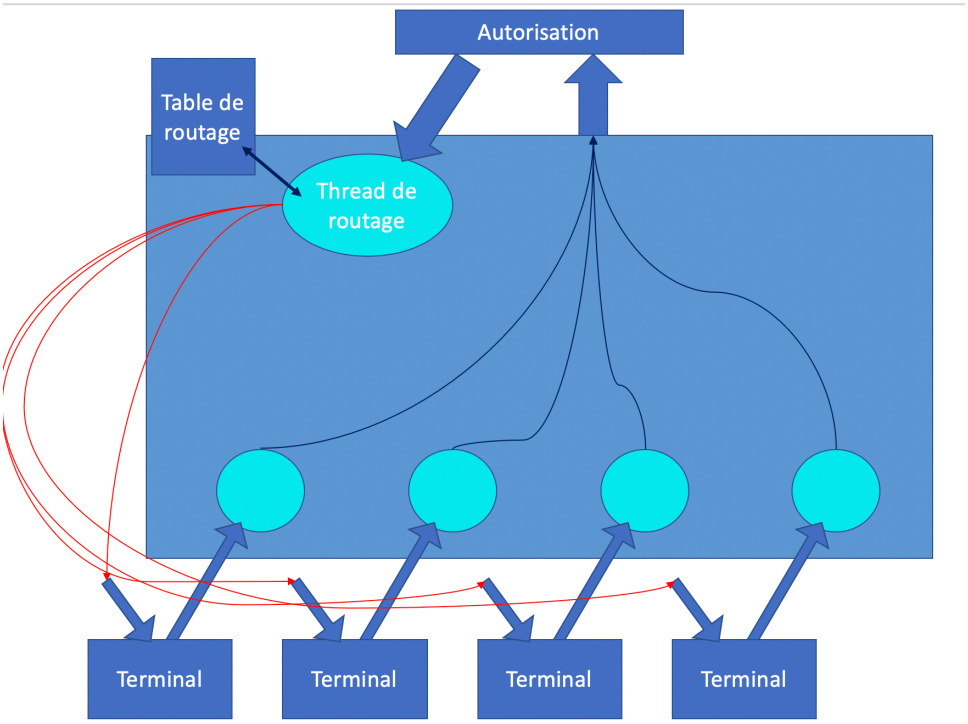
Ajout de l'utilisation d'un annuaire : nous avons eu accès plusieurs fichiers qui permettaient de créer un annuaire et de l'utiliser. Nous avons donc ajouté la fonction permettant de générer un annuaire aléatoire contenant un numéro de carte et un solde aléatoire. Dans le terminal nous avons donc retiré la lecture dans un fichier et nous sommes passés à la lecture dans un annuaire après génération de celui-ci. Il y avait dans les fichiers donnés des structures que nous pouvions utilisés afin de récupérer uniquement le numéro de carte par exemple.

Jour 5 :

Dans notre dernière version nous pouvons lancer plusieurs terminaux et nous avons l'impression que l'ordre des opérations est aléatoire tellement le programme s'exécute vite mais en réalité en l'absence de thread chaque opération se lance l'une après l'autre et dans l'optique où une personne du terminal 3 voudrait effectuer un retrait elle devra attendre qu'une opération soit effectuée sur les terminaux 1 et 2. Ceci à grande échelle n'est évidemment pas souhaitable.

Pour régler ce problème nous ajoutons les threads, chaque terminal est relié à un thread qui va s'occuper de transmettre la demande au serveur d'autorisation en écrivant dans un tube relié à celui-ci. En sortant du tube autorisation (après traitement) le message repasse par un thread, que nous avons appelé thread de routage, qui va regarder dans une table de routage afin de renvoyer le message au terminal correspondant. La table de routage est en fait un tableau composé du numéro de carte, du terminal à partir du quelle la demande a été émise et du descripteur de fichier permettant d'écrire dans le tube du terminal en question. Ainsi lorsqu'une réponse arrive dans le thread de routage, celui-ci parcourt la table de routage pour trouver le numéro de carte et y trouver le descripteur de fichier qui permet d'écrire dans le terminal qui a émis la demande.

Ci-dessous un schéma de la situation



En introduisant les threads nous introduisons aussi une section critique, il s'agit de la table de routage. En effet l'écriture et la lecture dans la table de routage, contenant les informations permettant au thread de routage d'envoyer une réponse vers le bon terminal, ne doit être faite que par un seul thread

à la fois, sinon nous pouvons nous retrouver à lire des informations supprimées par un autre thread au même moment. C'est pour cette raison que nous avons utilisé un sémaphore. Nous l'initialisons avec une seule place, ensuite lorsqu'un thread rentre des informations dans la table de routage il rentre en section critique et donc met en attente les threads qui veulent y accéder. De même lorsque le thread de routage veut consulter des informations dans la table de routage, il rentre en section critique et prend la place du sémaphore.

Suite à cela, notre programme fonctionne, il suffit de lui rentrer un nombre de terminaux et à l'exécuter.

III. Limitations de notre simulation et bugs rencontrés

Notre simulation ne possède pas de réseau interbancaire, néanmoins nous avons réfléchi à comment le mettre en place.
Chaque serveur d'acquisition possèderait le numéro de la banque à laquelle il est associé. Une première vérification serait faite en comparant le numéro de la banque du serveur d'acquisition avec celui du numéro de carte reçu en provenance d'un terminal, car les 4 premiers numéro correspondent à l'identifiant de la banque. Si le numéro de carte ne correspond pas à la banque du serveur d'acquisition, la demande serait directement envoyée dans un tube qui communique avec le serveur interbancaire. Pour ne pas bloquer tout le processus interbancaire, un thread s'occuperait de la lecture dans le tube. Une fois qu'une demande aura été lu, le numéro de carte de la demande sera découpé pour en extraire les 4 premiers chiffre, ensuite ces numéros seront comparés aux numéros stockés dans une table de routage. A chaque numéro de banque sera associé, dans la table de routage, un descripteur de fichier qui permettra d'écrire dans le tube de la banque concerné. Une fois la demande redirigé dans le bon tube de la bonne banque un thread dans le serveur d'acquisition de la banque se chargera de lire les demandes en provenance du serveur interbancaire et à les acheminer vers le serveur d'autorisation. Ce thread permettra de ne pas bloquer tout le processus acquisition de la banque car il ne lira que les demande provenant du serveur interbancaire. Une fois la demande autorisée ou non elle sera renvoyée dans un tube vers le serveur interbancaire. Une nouvelle fois, un thread dans le serveur IB s'occupera de lire la réponse et de la router vers le serveur acquisition dont elle provenait initialement. Pour que ce thread puisse router au bon serveur, lorsque que la demande avait été émise pour la première fois le numéro de carte ainsi que le serveur qui avait acheminé la demande auront été placé dans une autre table de routage qui permet donc au réponse (contenant le numéro de carte) d'être routé au bon endroit.
Bien évident pour que cela puisse fonctionner il faut que le serveur interbancaire créer les tubes et s'occuper de fork les serveurs d'acquisition.

Dans nos programmes, il serait judicieux de mieux allouer les tableaux, notamment avec des mallocs, ainsi que de libérer la mémoire, c'est une optimisation que nous pourrions rajouter.

Nous avons décidé de faire un tableau de routage en int, car nous pensions au début que cela serait plus facile. Mais les numéros de carte étant des longs, lors de leur conversion de character à int, nous nous retrouvons avec des nombres qui n'ont pas forcément de sens. Néanmoins cela ne pose concrètement pas de problèmes lors de la vérification des numéros de carte donc nous avons décidé de ne pas changer notre tableau de routage.

Chaque terminal n'émet qu'une seule demande. Néanmoins nous avons laissé la porte ouverte à une amélioration en laissant volontairement une boucle while dans chaque terminal de tel sorte à ce que les terminaux puissent émettre, avec une amélioration de notre code, plusieurs demandes. Par exemple on pourrait imaginer une demande toutes les 20 secondes.

Les numéros de carte sont pris aléatoirement dans un annuaire de clients, par conséquent plusieurs clients peuvent avoir le même numéro de carte et donc effectuer une demande de paiement simultanément sur 2 terminaux différents.

Enfin, nous ne décrémentons pas le solde des clients. Cette amélioration pourrait être intéressante à mettre en place. Pour chaque client nous pourrions décrémenter son solde associé et ainsi réécrire le nous solde dans le fichiers annuaires client que nous avons utilisé.

Nous avons également rencontré quelques problèmes du fait que nous ayons travaillé sur mac et non sous UNIX, certaines commandes étaient différentes (execlp, alea) et il a donc fallu ajuster notre code en conséquence.

IV. Annexe : Réponse aux questions

- 1/ Combien de processus seront créés dans le projet ?

Nous avons un processus pour le réseau interbancaire.
Ensuite, chaque banque possède un serveur d'acquisition et un serveur d'autorisation plus les terminaux appartenant à cette banque.
Donc pour chaque banque existante nous avons 1 processus pour le serveur d'acquisition, 1 autre pour le serveur d'autorisation et ensuite X processus en fonctions du nombre de terminaux de payement.
Dans le cas de l'exemple ci-dessous dans l'arbre des processus nous avons donc : 8 processus.

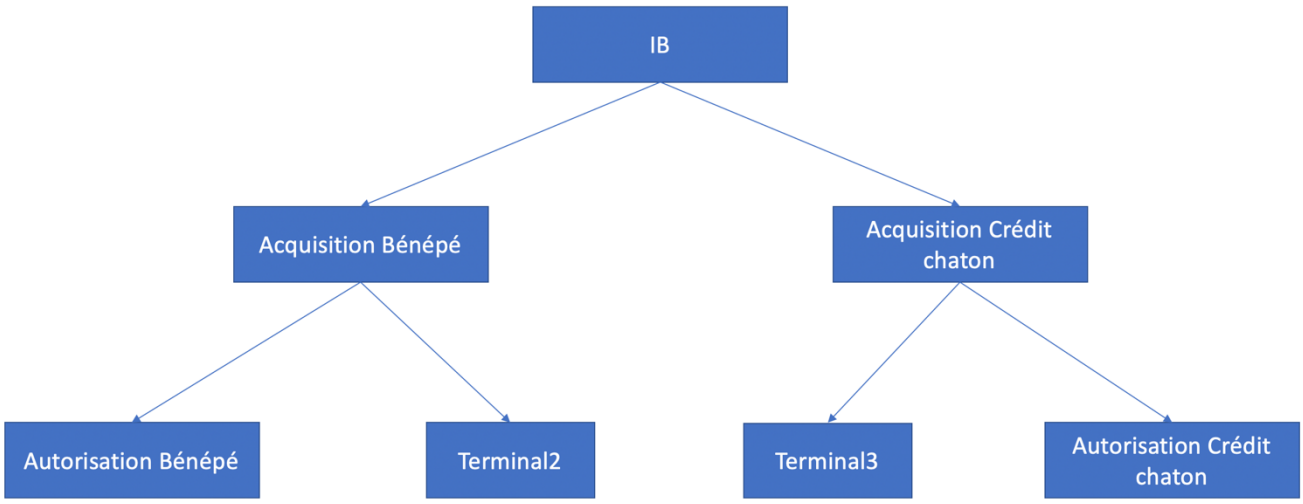
- 2/ Combien d'exécutables allez-vous programmer ? Est-ce que un seul exécutable est acceptable ?

Nous allons programmer 4 exécutables : un pour l'autorisation, un pour l'acquisition, un pour le réseau interbancaire et enfin un dernier pour les terminaux.
Il n'est pas souhaitable d'avoir un seul exécutable car, d'une part lorsqu'une modification est faite tout le programme doit être recompilé et ensuite comme nous sommes en binôme, travailler à deux sur un seul exécutable est assez difficile et amène forcément des erreurs de communication.

- 3/ Qui (quel processus) est créé par qui ?

Le réseau interbancaire créé les processus d'acquisitions qui à leur tour créer les processus des terminaux et les processus d'autorisation.

- 4/ Dessinez l'arbre des processus du projet.



- 5/ Combien de tubes sont nécessaires pour le projet ?

Les serveurs d'acquisitions ont besoins de pouvoir transmettre et recevoir les informations à destination du Terminal, mais également de pouvoir envoyer et recevoir des informations en direction du serveur d'autorisation et en direction du serveur Inter Bancaire. Et enfin le serveur interbancaire a besoin de tubes afin de recevoir et transmettre les informations aux banques concernées. Dans le cas de l'exemple avec le crédit Chaton et la banque Bénépé, si on considère un terminal par banque il faudrait 12 tubes. En outre, pour chaque banque, il faut 2 tubes par terminal crée, 2 tubes pour communiquer avec le serveur d'autorisation, et enfin 2 tubes pour communiquer avec le serveur interbancaire.

- 6/ Quels types de tubes allez-vous utiliser ?

Des tubes nommés unidirectionnels.

- 7/ Qui (quel processus) crée quels tubes ?

Les serveurs d'acquisition créer les tubes nécessaires à la communication avec les serveurs d'autorisations et avec les terminaux. Le serveur interbancaire créer les tubes pour communiquer avec les serveurs d'acquisition.

- 8/ Dessinez vos processus et vos tubes.