Rapport de projet

Ce document a pour but de décrire le déroulement de notre projet dans l'unité d'approfondissement informatique en système d'exploitation. Ce projet nous a permis de mettre en œuvre une simulation de système de paiement par carte bancaire.

Ce rapport contient l'ensemble des éléments du projet. Nous présenterons, dans un premier temps notre projet d'un point de vue technique avec le cahier des charges fonctionnelles puis les spécifications et solutions techniques. Nous aborderons ensuite le fonctionnement de notre projet ainsi que les solutions choisis pour la résolution de chaque problème. Pour terminer, nous présenterons nos impressions sur le projet concernant les difficultés techniques rencontrées et les perspectives ouvertes.

Dans un second temps, nous présenteront la manière dont nous avons géré le projet et comment il a été découpé en tâches simples afin de répartir le travail.

Enfin nous verrons ce que le projet nous a apporté : d'un point de vue technique, gestion de projet, pédagogique, puis personnel.
Nous espérons que vous prendrez autant de plaisir à lire ce rapport que nous en avons pris durant tout le déroulement de ce projet.

Roger et Théophane

Sommaire

Introduction

Contexte :

L'objectif du projet est de simuler les échanges entre banques permettant à un particulier de payer ses achats avec sa carte bancaire (dite "carte bleue"), même si celle-ci n'est pas émise par la même banque que celle du vendeur.

[Simulation d'un système de paiement par carte bancaire, Jean COUSTY & Laurent NAJMAN, 2019, p.4]

Dans le cadre de ce projet nous retrouverons quatre unités: Le *serveur Terminal* qui permet au client d'effectuer une demande de transaction, le *serveur acquisition* qui permet de rediriger la demande de débit en fonction de la banque auquel appartient le client, le *serveur interbancaire* qui reçoit la requête émise par le serveur acquisition qui redirige la demande vers la banque cliente et enfin le *serveur autorisation* qui reçoit une demande du serveur interbancaire puis vérifie le solde client pour émettre une réponse en effectuant le chemin inverse.

Les quatre entités ont ainsi été traitées de façon indépendante afin de travailler par palier au cours de ce projet et isoler les parties fonctionnelles, des parties en cours de débogage. Vous pourrez ainsi retrouver dans la suite de ce rapport des explications organisées par entité.

Rapport Technique

1. Cahier des charges fonctionnelles

Le *terminal* est l'entité qui permettra d'envoyer une demande de paiement au serveur acquisition. Il enverra une demande sous la forme :

|Numéro de carte bancaire*|Type de message|Valeur***|**

*Codé sur 16 chiffres
**Demande/Réponse
***S'il s'agit d'une demande alors le champ correspondra à la valeur du montant de la transaction sinon pour une réponse il vaudra 1 si la réponse est acceptée et 0 si elle ne l'est pas.

Dans ce projet le terminal prendra aléatoirement un numéro de carte dans une liste de cartes bleues et générera un montant aléatoire pour effectuer une demande de paiement. Le terminal devra ensuite se mettre en attente pour réception du message de réponse à sa requête.
La ressource nécessaire au programme sera alors le fichier qui recense toutes les cartes existantes.

Le *serveur d'acquisition* est une sorte de routeur. Il doit pouvoir accepter des messages de toutes les entités, étant donné l'architecture en étoile de notre projet. Ainsi il doit être capable de diriger des demandes provenant de terminaux, soit vers leur banque si le terminal et la carte bleue sont émis par la même banque ou bien vers le serveur interbancaire si ce n'est le cas. Sa deuxième mission est de réceptionner les réponses provenant du serveur d'autorisation ou du serveur interbancaire et de rediriger chaque message vers leur terminal correspondant.
Il doit pouvoir accéder au fichier listant les cartes bleues filtrées par banque pour effectuer son routage correctement.

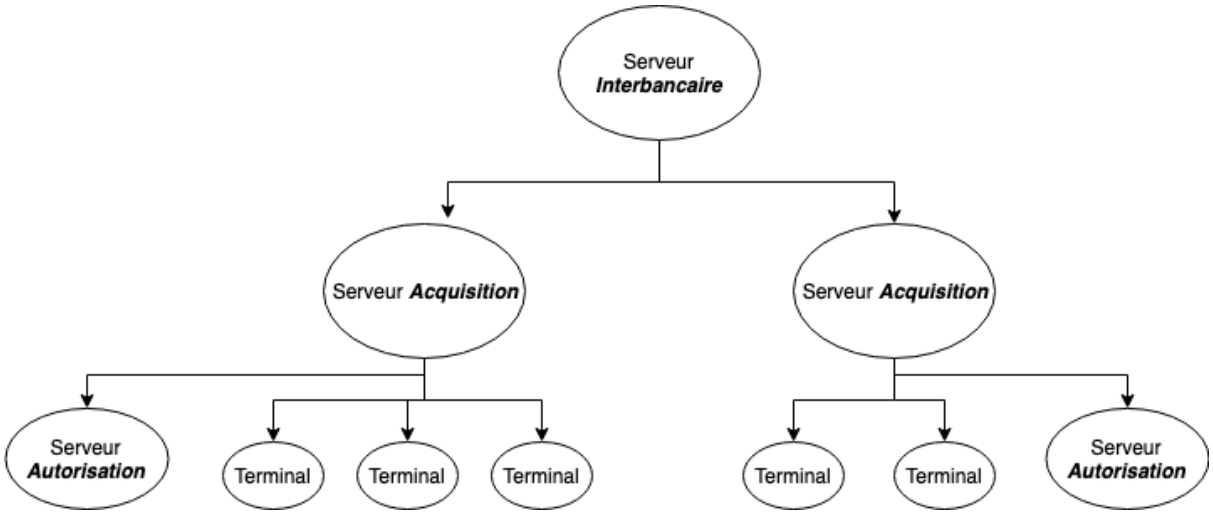
Le serveur autorisation est l'entité qui autorise ou non le paiement. Dans un premier temps il effectue une vérification sur l'existence de la carte dans sa "base de données". Il effectue ensuite une vérification sur le compte du client afin de voir si son compte est suffisamment approvisionné pour le paiement et envoie une réponse sous la forme suivante :

|1456901256874619|Reponse|1|

Le programme doit donc avoir accès au fichier filtré de cartes bleues, contenant le solde du client, le client correspondant au numéro de compte lors de la transaction.

Le serveur interbancaire est le second routeur du projet. Il accepte les messages en provenance des serveurs d'acquisition. Il oriente d'abord chaque message de demande vers le serveur acquisition correspondant à la banque du client, identifié par les 4 premiers chiffres de la carte bancaire. Enfin il accepte les messages de réponse pour les diriger vers le terminal ayant effectué la demande initiale.
Il doit accéder au fichier de carte bleue filtré par banque pour effectuer son routage.

2. Spécifications techniques



Arbre de création des processus

Lors de ce projet un processus sera créé pour simuler le serveur interbancaire, un autre processus pour les serveurs d'autorisation et d'acquisition et ceci pour chaque banque. Enfin il y a un processus par terminal.

Pour l'exécution de ce projet, nous aurons besoin de créer 4 exécutables, un pour les serveurs d'autorisation, un autre pour les serveurs d'acquisition, un pour le serveur interbancaire et un dernier pour les terminaux. Le projet peut être réalisé dans un seul et même exécutable, néanmoins plusieurs problèmes apparaissent. Dans un premier temps la gestion en binôme devient laborieuse puisqu'il faut travailler sur le même fichier sans se « déranger ». De plus, plus le projet grandit plus la probabilité d'erreur est forte et il est nécessaire de découper le projet en plus petits sous-projets, nous permettant ainsi de pouvoir les tester séparément sans à avoir la totalité du programme. Enfin il existe également des problèmes sécuritaires lorsque toutes les données sont centralisées dans un seul et même fichier, en effet le cloisonnement des données n'est pas bien mis en place.

Le processus initial est le processus interbancaire, effectivement nous avons fait le choix de le placer au centre de notre architecture (voir schéma ci-après), ensuite ce serveur va créer un processus pour chaque banque de son réseau. Une banque étant composée d'un serveur d'acquisition, qui constitue le lien avec le réseau interbancaire, ainsi qu'un serveur d'autorisation qui est créé par le serveur d'acquisition de sa banque. Enfin à des fins de simulations le processus des terminaux est créé par le serveur d'acquisition de la banque émettrice du terminal.

Les tubes que nous allons utiliser sont des tubes non nommés, effectivement un paiement nécessite d'être sécurisé. Les tubes anonymes ne sont pas des fichiers à part entière que l'on peut retrouver dans notre système comme les tubes nommés ils garantissent ainsi une sécurité pour nos communications.
Seulement les tubes non nommés ne sont pas bidirectionnels et ne permettent pas une communication dans les deux sens comme peuvent le faire les tubes nommés. Il sera donc nécessaire de créer une paire de tube pour chaque entité souhaitant communiquer.

Les « tuyaux » permettant la communication entre le réseau interbancaire et les serveurs d'acquisition sont créés par le processus du réseau interbancaire. De même que les serveurs d'acquisition créés les « tuyaux » de communication avec les terminaux et les serveurs d'autorisation qui leurs sont propres.

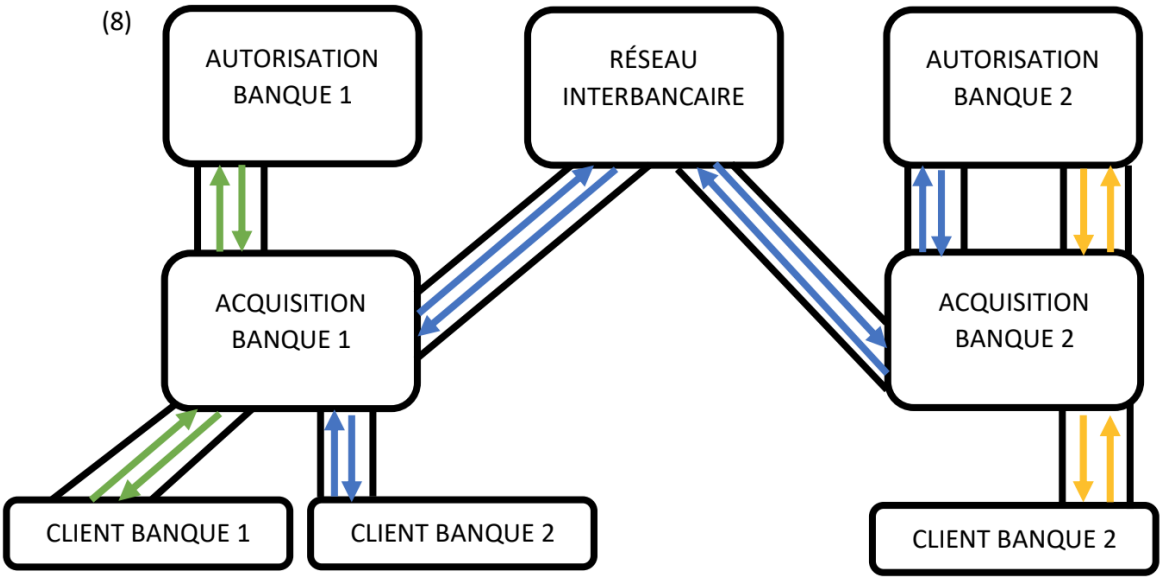


Schéma organisationnelle de la communication interprocessus

3. Programmes (manuel technique)

Dans un premier temps nous allons détailler le programme utilisé pour créer un terminal ainsi que celui pour créer un serveur autorisation. Effectivement, leur architecture globale est très similaire puisque l'une est l'inverse de l'autre.

Le terminal est un exécutable prenant 2 paramètres lorsqu'on l'appelle, le premier est le "tuyau" duquel il envoie les données (la sortie) et le deuxième paramètre correspond au "tuyau" par lequel le terminal reçoit les informations (l'entrée). Nous généralisons ici la notion de paramètre mais en ce qui concerne les "tuyau" c'est l'adresse de celui-ci qui est transmis, permettant ainsi de pouvoir l'utiliser. Il en est de même pour l'exécutable pour le serveur autorisation, mis à part que l'information suit le chemin inverse.
Pour le terminal nous faisons appelle aux fonctions fournies lors de ce projet : message.c, alea.c ; mais également aux deux fonctions de lecture et d'écriture que nous avons vu lors de l'unité de Système d'Exploitation contenues dans le fichier fichier.c. Nous verrons par la suite que ces fonctions sont utilisées pour toutes communications à travers un tube de manière générale. Nous utilisons ces fonctions pour pouvoir simuler une demande de transaction respectant les caractéristiques vues précédemment. Pour ce faire on fait appelle à la fonction alea() pour pouvoir avoir un montant mais aussi aux numéros de compte contenus dans la base de données du projet et enfin à la fonction message pour créer la demande au format universel.

À des fins de tests permettant de simuler plusieurs demandes nous avons créé une fonction decoupe3() dans le fichier message.c fourni, elle est très similaire à la fonction de base mais elle s'adapte au format de ligne que nous voulons découper. En effet, nous voulons être sûr que la demande s'effectue d'un compte existant, nous nous basons donc sur le fichier "annuairefiltre.an" (quand nous travaillons que sur une banque) et comme ce fichier contient pour chaque ligne le numéro de compte, le numéro de la banque ainsi que le solde nous avons revu la découpe.

Ensuite le processus va écrire dans le "tuyau" à destination de son serveur acquisition que nous verrons par la suite le message, puis va lire dans le "tuyau" entrant la réponse que le serveur va lui fournir. La fonction litLigne() sert également de "wait" puisqu'elle suspend le processus jusqu'à ce que celui-ci reçoive la réponse. Une fois que la réponse est reçue, elle est traitée en fonction de la valeur du message qui a été découpé au préalable, pour afficher la réponse adéquate.

Par opposition le serveur autorisation, procède de la même manière mais dans le sens inverse. C'est-à-dire que le processus attend la demande de transaction puis la traite puis renvoie la réponse, en fonction du solde du client au format universel à l'aide des fonctions fournies, au serveur acquisition correspondant.

Serveur acquisition :

Dans un premier temps nous allons décrire un serveur autorisation utilisant la duplication de processus (fork()), ensuite nous expliquerons l'amélioration apportée avec l'ajout des threads.

Le premier exécutable créé se compose d'une seule fonction principale prend en paramètre la valeur de la banque à des fins de test, puisque nous commençons en testant avec une seule banque. Comme dit précédemment dans l'arbre des processus nous avons choisi une disposition en étoile, avec le serveur acquisition étant à la base du serveur autorisation et des terminaux lui correspondant, il nous faut donc créer ces connexions. De plus nous voulons simuler plusieurs terminaux, chacun possédant sa propre connexion, nous avons dans un premier temps opter pour une exécution séquentielle assurant le fait qu'il n'y ait pas de conflit dans les "tuyaux". Ainsi l'initialisation de la connexion avec le serveur autorisation se fait en dehors d'une boucle qui va être exécutée un nombre de fois correspondant au nombre de demande que nous voulons simuler. Cette boucle va contenir une double duplication, une première pour pouvoir exécuter le serveur autorisation et une deuxième pour pouvoir exécuter le terminal. Pour ce faire on utilise la fonction fork() et l'on fait appel aux exécutables grâce à la fonction exec() que l'on va utiliser dans le processus fils de chaque duplication.

Le deuxième exécutable utilise les threads, nous allons voir les ajouts par rapport à la version précédente mis en place.

```
struct Request {
  int ID;
  int banque;
  int pipeTerminalUp[2];
  int pipeTerminalDown[2];
};
```

Comme la capture ci-dessus nous avons rajouté une structure, ce qui permettra par la suite d'identifier chaque thread mais également de lui notifier les adresses du terminal avec lequel il doit communiquer.

Ensuite comme nous utilisons des threads il faut rajouter une fonction qui sera exécutée par tous les threads que nous allons créer. C'est dans cette fonction que nous allons retrouver les lectures et écritures vues précédemment mais aussi la duplication à l'intérieur même du thread pour pouvoir exécuter un terminal avec les adresses des "tuyaux" que nous retrouvons dans la structure.
Dans la fonction principale, nous avons, contrairement à la version précédente, un tableau de structure pour chaque demande et donc thread que nous allons créer. Ensuite pour pouvoir simuler un serveur d'autorisation opérationnelle en permanence, nous avons placé la duplication identique à celle de la version précédente dans une boucle infinie. Au préalable nous avons créé tous les threads nécessaires aux tests en utilisant la fonction pthread_create(), puis nous avons initialisé la sémaphore définie de manière globale, pour qu'elle puisse être accessible par tous les threads.
Cet sémaphore va garantir qu'il n'y a pas de modification du fichier bancaire en même temps et ainsi qu'il n'y a pas de conflit. Pour ce faire nous effectuons un sem_wait en début de fonction du thread et un sem_post en fin de fonction, de ce fait si un thread est déjà en cours d'exécution alors aucun autres ne peuvent être exécuter.

Pour la suite des explications, nous aborderons le côté théorique que nous voulions mettre en place mais que nous n'avons pas pu implémenter.

Après avoir réalisé cette version du serveur acquisition, nous nous sommes rendu compte qu'il comportait des défauts, le principal étant la boucle infinie puisqu'elle bloque le programme indéfiniment mais aussi parce qu'une erreur lors d'une transaction bloque tout le système.
Ensuite pour l'implémentation du serveur interbancaire nous avons pensé au même système que le serveur acquisition créant son serveur acquisition et les terminaux. Néanmoins des différences subsistent puisque ce serveur est situé au centre du dispositif et possède de ce fait un routage beaucoup plus complexe à gérer que celui du serveur acquisition.
Comme pour les routeurs utilisés dans nos réseaux, nous avons pensé à l'implémentation d'une table de routage contenant les adresses des serveurs d'acquisition associés à la banque correspondante. Ainsi le message envoyé par le serveur acquisition contenant la banque à laquelle il doit être transmise pourra sans aucun souci être traité. Cependant il existe également un problème de redirection quand le message a été traité par le serveur autorisation lui correspondant et qu'il doit être renvoyé vers le bon terminal. Pour pallier à ce problème, rappelons-nous que le serveur acquisition correspondant au terminal émetteur sait vers quel terminal envoyer, or le serveur interbancaire sait vers quelle serveur acquisition il doit renvoyer le message, ainsi le message peut refaire tout le chemin inverse.

4. Difficultés techniques

Le terminal a été le premier point traité et nous a posé quelque problème concernant sa réalisation. Les premières versions du terminal utilisaient des tubes nommés et n'affichaient pas les résultats des opérations sur la sortie standard. En effet nous avions créé une version du terminal qui ne recevait rien en argument et qui ne renvoyait rien sur sa sortie standard. Il s'agissait d'une version naïve du terminal qui affichait grâce à un printf des phrases. En tentant d'avancer nous nous sommes retrouvés face à un mur ne sachant pas faire communiquer de façon simple un autre processus avec notre version du terminal hermétique. C'est pourquoi suite à une concertation avec notre professeur nous sommes partis sur la réalisation d'un terminal qui utilisait des arguments correspondant au descripteur de fichiers sur lequel le programme devait écrire et lire.

La mise en place du serveur acquisition a été laborieuse notamment concernant l'implémentation des fonctions de bibliothèque des comptes clients. Cette tâche a nécessité de comprendre et tester les différentes options fournies et de déboguer.

La mise en place de la communication interprocessus avec des tubes non nommés a été une tâche ardue de par la compréhension du fonctionnement et l'adaptation au projet. La création de pair de tube et la synchronisation des processus ont donc été mis en place à cet effet afin de répondre à cette difficulté de faire communiquer deux processus.