Simulation d'un système de paiement par carte bancaire

Mini-projet PR-3602 - 2019

Sujet rédigé par Jean Cousty et Laurent Najman,

très librement inspiré d'un projet de Jérome Gueydan pour l'ENSTA

5 avril 2019

Table des matières

Avertissement

Le projet suivant est à faire par groupe de deux étudiants (en une semaine entièrement dédiée au projet, plus un **petit** travail personnel si nécessaire), et à rendre le mardi 3 mai 2016 avant 9H.

Pour faire court, sachez que tous les projets "semblables" auront une note de zéro, et que les auteurs seront convoqués en conseil de discipline. Ceux qui arriveront à faire la preuve qu'ils ont écrit le projet duquel les autres se sont inspirés n'auront aucune pénalité supplémentaire par rapport à la note de zéro. Si vous faites le projet à plus de deux, un coefficient de réduction proportionnel sera appliqué. Par exemple, si vous faites le projet à trois, la note sera réduite d'un tiers. Aucun bonus ne sera donné pour un étudiant travaillant seul.

Plagiat

Le plagiat est l'utilisation, sans citation appropriée, du travail intellectuel d'une autre personne dans un travail soumis à évaluation. Ce qui suit est très fortement inspiré et traduit de la page

http ://www.csd.abdn.ac.uk/teaching/handbook/both/info.php ?filename=cheating.txt.

Quand vous écrivez un rapport ou du code qui contient des parties ou qui paraphrase le travail d'autres personnes, vous devez clairement le signaler. En particulier, les citations doivent être données entre guillemets, avec les références appropriées.

1. Le code soumis pour évaluation doit clairement être annoté avec le nom de l'étudiant qui a soumis l'exercice, avec la date de rédaction.
2. Quand un exercice contient du code qui n'a pas été écrit par l'étudiant qui soumet le travail, ou contient du code écrit par l'étudiant lui-même à un autre moment, le code en question doit être clairement identifié et annoté avec le nom de l'auteur, le copyright (si différent), la date d'achèvement ou de publication, et une référence à la source (par exemple une URL ou une référence bibliographique).
3. En principe, la discussion avec d'autres étudiants du contenu du cours et des exercices non-évalués est encouragée, car une telle discussion amène généralement à une meilleure compréhension du sujet. Cependant, les discussions doivent rester à un niveau général, et ne doivent pas descendre à un niveau détaillé de conception ou de codage. Le travail soumis **pour évaluation doit être un travail individuel**. Si vous soumettez un travail produit conjointement avec une autre personne, ou qui inclus le travail de quelqu'un d'autre, **ceci doit être clairement indiqué, et le code en question doit être clairement identifié**. L'évaluation du travail se fera sur la base de la valeur ajoutée par l'étudiant.
4. De manière identique, bien que nous encouragions la réutilisation de logiciel existant comme une bonne pratique d'ingénierie Logicielle, les origines d'une telle réutilisation doivent **être clairement indiquées, et le code identifié**. L'évaluation du travail se fera sur la base de la valeur ajoutée par l'étudiant.
5. Quand le travail est à réaliser en groupe de deux étudiants, le rapport écrit ou le travail doit clairement identifier et distinguer ce qui a été réalisé par un des étudiants, ou ce qui a été fait par le groupe en entier.

Modalité d'évaluation des compétences acquises

— Vous devez mettre à jour votre **dépôt svn** [1] avec la dernière version de votre projet avant mardi 3 mai à 9H. A cette date votre dépôt devra contenir le code qui sera évalué (sans les fichiers exécutables ou objet), le makefile, le rapport du projet sous format électronique '.pdf' et le mode d'emploi. Le rapport papier doit être déposé le même jour au secrétariat du Département Informatique/Télécom (bureau 5251). Si votre dépôt svn

1. Voir la section 5.1 en annexe pour plus de détails sur le dépôt svn qui a été créé pour vous à l'ESIEE.

n'est pas à jour à cette date, un barème de "moins deux points" par jour de retard sera appliqué.
— Le projet devra se compiler en exécutant la commande make sur une machine tournant Linux. **Un projet qui ne respectera pas cette consigne ne pourra pas être corrigé.**
— Le rapport du projet décrira les diverses solutions aux différents problèmes rencontrés, et **justifiera** la solution choisie.
— Votre évaluation dépendra de la qualité du rapport, et de la qualité du code.
— Les compétences à acquérir (et donc l'évaluation) comprennent une partie liée à la rédaction et à la justification des choix retenus, et également une partie liée à l'écriture du code correspondant aux choix que vous aurez faits. Vous pourrez donc valider l'acquisition de compétences si vous n'avez pas programmé vos propositions, mais vous n'aurez pas forcément validé l'acquisition de toutes les compétences si le code que vous avez écrit n'est pas proprement justifié.
— Enfin une **évaluation orale individuelle** pourra avoir lieu à n'importe quel moment de la semaine dans les créneaux et salles réservés sous ADE au projet PR-3602.

Remarque

L'énoncé peut sembler épais de prime abord. Ce n'est pas le cas : il se résume aux pages 4 à 11. Le reste du document a pour objectif de vous aider dans la réalisation du projet, en indiquant des éléments de solutions, en attirant l'attention sur les points essentiels, et en donnant une démarche saine de progression.

La rédaction du texte du sujet est inspiré par un projet anciennement proposé par Jérome Gueydan pour l'ENSTA et bénéficie d'une expérience de plusieurs années à l'ESIEE.

1 Introduction

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.

Avant de présenter le sujet, examinons le fonctionnement du paiement par carte bancaire.

1.1 Le principe du paiement par carte bancaire

Le paiement par carte bancaire met en relation plusieurs acteurs :
— le client, qui souhaite régler un achat avec la carte bancaire qu'il possède et qui lui a été fournie par sa banque (le Crédit Chaton) ;

— le commerçant, qui est équipé d'un terminal de paiement fourni par sa propre banque (la Bénépé) ;
— la banque du commerçant (la Bénépé) à laquelle est connectée le terminal de paiement ;
— la banque du client (le Crédit Chaton) qui va dire si la transaction est autorisée (donc si le compte de son client est suffisamment provisionné ou non).

Le terminal du commerçant est relié à la banque Bénépé grâce à une simple ligne téléphonique. La banque Bénépé est connectée à toutes les autres banques installées en France, et notamment au Crédit Chaton, grâce à un réseau dédié : le réseau interbancaire (voir fig. 1).

Supposons maintenant que le client lambda se rend chez son revendeur de logiciels préféré pour acheter la toute dernière version du système d'exploitation "FENÊTRES". Au moment de passer en caisse, il dégaine sa carte bancaire, le caissier l'insère dans son terminal de paiement et le client doit, après avoir regardé au passage la somme qu'il s'apprête à débourser, saisir son code confidentiel.

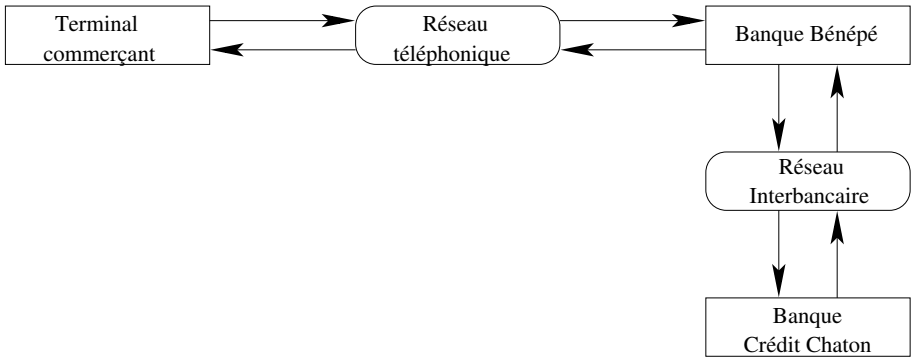

FIGURE 1 – Principe de communication des demandes d'autorisation et des différents réseaux

Ce code est directement vérifié par la carte (plus exactement par la puce contenue dans la carte). Si le code est erroné, la transaction s'arrête là. Si le code est bon, en revanche, les opérations suivantes ont lieu :

1. Le terminal se connecte (via le réseau téléphonique) au serveur de la banque Bénépé et envoie le numéro de la carte bancaire ainsi que le montant de la transaction.
2. Le serveur de la banque Bénépé regarde le numéro de la carte et, se rendant compte qu'il ne s'agit pas d'une des cartes qu'il a émises, envoie le numéro de carte avec le montant de la transaction au serveur de la banque Crédit Chaton, via le réseau interbancaire permettant de relier les différentes banques.

3. Le serveur de la banque Crédit Chaton prend connaissance du numéro de la carte bancaire et vérifie que le compte correspondant à ce numéro dispose d'un solde suffisant pour honorer la transaction.
4. Si c'est le cas, il répond à la banque Bénépé (toujours via le réseau interbancaire) que le paiement est autorisé. Si ce n'est pas le cas, il répond le contraire.
5. Enfin, le serveur de la banque Bénépé transmet la réponse au terminal du commerçant.
6. La transaction est validée ("paiement autorisé") ou refusé ("paiement non autorisé").

1.2 La demande d'autorisation

La suite des opérations décrites ci-dessus se nomme la "demande d'autorisation" et a essentiellement pour but de vérifier que le compte client est bien provisionné (ou qu'il a une autorisation de découvert). Cette suite d'opération montre que le rôle des banques est double. En effet, l'action de la banque diffère selon si la demande d'autorisation est locale ou non. i) Si ce n'est pas le cas, son rôle est d'acheminer la demande vers la banque qui a la capacité de la traiter (*cf.* étape 2 ci-dessus). ii) Si la demande d'autorisation est locale, son rôle est de vérifier que celle-ci peut être honorée (*cf.* étape 4 ci-dessus) et de produire une réponse qui est ensuite acheminé vers le terminal du commerçant depuis lequel la demande a été émise.

Chaque banque est donc composée de deux serveurs.

Le serveur d'acquisition. Il s'agit du serveur de la banque du commerçant auquel se connecte le terminal via le réseau téléphonique. Une fois connecté, le terminal envoie au serveur d'acquisition toutes les informations concernant la transaction, notamment le montant, le numéro de carte et des données permettant d'assurer la sécurité de la transaction. Le serveur d'acquisition a ensuite pour rôle d'acheminer les données vers un autre serveur capable de les traiter.

Le serveur d'autorisation. Il s'agit du serveur de la banque du client auquel le serveur d'acquisition transmet la demande d'autorisation de paiement émise par le terminal. C'est lui qui est chargé de produire la réponse à la demande d'autorisation. Cette réponse suit donc le chemin inverse, c'est à dire : serveur d'autorisation de la banque du client → serveur d'acquisition de la banque du commerçant → terminal du commerçant.

1.3 Le routage

Pour effectuer le routage des demandes d'autorisation, c'est-à-dire pour déterminer à quelle banque chaque demande d'autorisation doit être transmise, le serveur d'acquisition utilise les premiers numéros de chaque carte bancaire concernée : ceux-ci indiquent la banque ayant émis cette carte. Dans ce projet, nous partirons des principes suivants :

— un numéro de carte est constitué de seize chiffres décimaux ;

— les quatre premiers correspondent à un code spécifique à chaque banque ;
— les serveurs d'acquisition des banques sont directement reliés au réseau interbancaire.

Chaque serveur d'acquisition analyse donc le numéro de la carte qui figure dans la demande d'autorisation qu'il reçoit, puis :

— si le client est dans la même banque que le commerçant (et que le serveur d'acquisition), il envoie la demande directement au serveur d'autorisation de cette banque ;
— si le client est dans une autre banque, le serveur d'acquisition envoie la demande sur le réseau interbancaire, sans se préoccuper de la suite du transit.

Le réseau interbancaire n'est donc pas un simple réseau physique : il doit aussi effectuer le routage des demandes d'autorisation, c'est-à-dire analyser les demandes qui lui sont fournies, envoyer chaque demande vers le serveur d'acquisition de la banque correspondante et, enfin, prendre en charge la transmission de la réponse lorsqu'elle lui revient.

2 Cahier des charges

L'objectif de ce projet est de simuler les mécanismes décrits ci-dessus, c'est-à-dire :

— le terminal envoyant une demande d'autorisation au serveur d'acquisition de sa banque ;
— le serveur d'acquisition effectuant le routage de la transaction vers le bon serveur d'autorisation et effectuant le routage des réponses qu'il reçoit en retour des terminaux ;
— le réseau interbancaire auquel sont connectés les différents serveurs d'acquisition, capable d'effectuer le routage des demandes et des réponses relayées par les serveurs d'acquisition ;
— le serveur d'autorisation fournissant la réponse à la demande d'autorisation ;
— le tout permettant un maximum de parallélisme.

Remarque. Cet exercice est avant tout "académique", mais n'est pas dénué d'intérêt ; en effet, des sociétés commercialisent toutes sortes de simulateurs pour tester le fonctionnement des nouveaux composants des systèmes monétiques, afin de valider leur fonctionnement avant leur mise en production.

Le cahier des charges fonctionnelles précise l'architecture générale, les fonctionnalités devant être programmées ainsi que les contraintes fonctionnelles à respecter. Le cahier des charges techniques fournit les restrictions concernant la mise en œuvre.

2.1 Cahier des charges fonctionnelles

2.1.1 Architecture fonctionnelle

Le schéma 2 précise celui qui a été présenté plus haut et retranscrit la description que nous avons fournie ci-dessus.

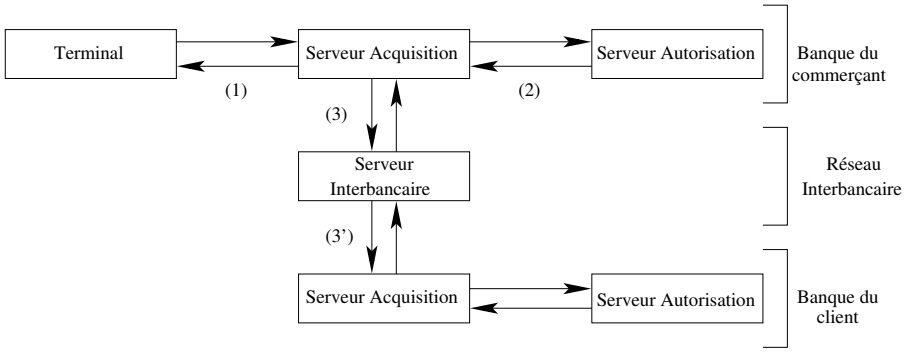

FIGURE 2 – Architecture fonctionnelle du projet

Chaque terminal est relié via le réseau téléphonique (1) au serveur d'acquisition de la banque du commerçant. Celui-ci est connecté au sein de la banque (2) au serveur d'autorisation de cette même banque.

Le réseau interbancaire comprend un serveur interbancaire qui est relié (3,3') aux serveurs d'acquisition des différentes banques. Toutes les autres banques de la place sont également reliées au serveur interbancaire, mais ne sont pas représentées sur ce schéma.

2.1.2 Terminal

Dans le cadre de ce projet, il n'est pas question d'utiliser de vrais terminaux ou de vraies cartes. Un terminal étant un moyen d'envoyer aux programmes des demandes d'autorisation, il sera simulé pour ce projet par un exécutable qui enverra un numéro de carte aléatoirement tirée dans la liste des cartes existantes et un montant de transaction aléatoire. Dans le système bancaire, chaque carte bleue est identifié par un code à 16 chiffres.

Pour fonctionner, un terminal a donc besoin de connaître la liste des numéros des cartes bleues existantes.

Chaque terminal envoie les informations générées vers son serveur d'acquisition, attend la réponse du serveur d'acquisition et affiche cette réponse à l'écran (paiement autorisé ou non).

Les échanges entre les terminaux et leur serveur ont lieu suivant un protocole bien déterminé : les informations sont formatées d'une certaine façon. Afin de simplifier le projet et de garantir l'interopérabilité des différents projets entre

eux (voir en annexe pour une explication de ce point), un seul protocole de communication est utilisé. Celui-ci est décrit en annexe de ce document.

2.1.3 Serveur d'acquisition

Un serveur d'acquisition n'a qu'une fonction de routage.

— Il doit pouvoir accepter des demandes d'autorisation provenant de terminaux et du réseau interbancaire.
— Il doit pouvoir effectuer le routage des demandes d'autorisation vers le serveur d'autorisation de la banque ou bien vers le réseau interbancaire.
— Il doit pouvoir accepter les réponses provenant du serveur interbancaire ou du serveur d'autorisation de la banque.
— Il doit pouvoir envoyer les réponses vers le réseau interbancaire ou le terminal (en étant capable d'apparier chaque réponse à la demande initiale).

Un serveur d'acquisition doit donc être capable d'utiliser le protocole de communication employé par les terminaux, ainsi que le protocole du réseau interbancaire (voir les protocoles définis en annexe).

Afin d'effectuer correctement le routage des messages, le serveur d'acquisition analyse, pour chaque message, le numéro (codé sur 16 chiffres) de la carte bleue concernée par le message : les 4 premiers chiffres indiquent la banque qui détient le compte associé à la carte.

Afin d'effectuer correctement le routage des réponses, un serveur d'acquisition doit garder trace des cartes insérées dans les terminaux de paiements et attendant une autorisation de prélèvement.

2.1.4 Serveur d'autorisation

Le serveur d'autorisation doit être capable de fournir une réponse à une demande d'autorisation. Pour fonctionner, le serveur d'autorisation doit donc avoir accès aux soldes des comptes des clients de la banque référencés par numéros de carte.

Dans le cadre de ce projet, nous utilisons une méthode simple : le serveur d'autorisation possède la liste des numéros de cartes émises par la banque, auxquels sont associés les soldes des comptes adossés à ces cartes ; lorsqu'une demande d'autorisation lui parvient, le serveur vérifie que le numéro de carte figure bien dans sa liste. Il contrôle alors que le solde du compte est suffisant pour effectuer la transaction, si c'est le cas il répond oui, sinon il répond non.

2.1.5 Le serveur interbancaire

Le serveur interbancaire n'a qu'une fonction de routage :

— Il doit pouvoir accepter les messages (demande d'autorisation et réponse à une demande d'autorisation) provenant des serveurs d'acquisition.
— Il doit pouvoir analyser le contenu des messages pour déterminer vers quel serveur d'acquisition il doit les envoyer.

Pour fonctionner, le serveur interbancaire doit posséder la liste des codes de 4 chiffres figurant en entête des cartes et les banques associées.

2.2 Cahier des charges techniques

2.2.1 Contraintes générales

— L'ensemble de la simulation tournera sur une seule machine.
— Les programmes seront écrits en langage C, et mettront en œuvre les concepts et les techniques apprises pendant le cours de Systèmes d'exploitation.
— Un maximum de parallélismes est exigé dans ce projet.

2.2.2 Mémoire des serveurs

Pour gérer les messages en attente, les serveurs d'acquisition et le serveur interbancaire ont une mémoire finie. La taille de cette mémoire (le nombre de case du tableau) sera précisée sur la ligne de commande au lancement d'un serveur.

2.2.3 Nombre de processus

Chaque composant "fonctionnel" tel qu'il a été présenté dans le cahier des charges fonctionnelles correspond à un processus. On trouve donc un processus pour le terminal (que l'on nomme *Terminal*), un processus pour un serveur d'acquisition (*Acquisition*), un processus pour un serveur d'autorisation (*Autorisation*) et un processus pour le serveur interbancaire (*Interbancaire*).

La simulation pouvant mettre en jeu plusieurs terminaux et plusieurs banques, on trouve en fait un processus *Terminal* par terminal, un processus *Acquisition* par banque et un processus *Autorisation* par banque.

Pour favoriser le parallélisme, chaque processus peut (si besoin est) être découpé en plusieurs threads.

2.2.4 Paramètres

Les paramètres nécessaires au fonctionnement des différents processus sont fournis via "la ligne de commande", à l'exception des paramètres suivants qui sont fournis sous forme de fichier :

— La liste des banques et leur code à 4 chiffres.
— La liste des cartes bancaires de chaque banque et le solde du compte associé.

Ces fichiers sont au format texte, chaque ligne contenant les informations demandées (code sur 4 chiffres et banque associée ou numéro de carte et solde du compte), séparées par un espace.

2.2.5 Gestion des échanges par tuyaux

Chaque terminal est connecté à son serveur d'acquisition par une paire de tuyaux. Le serveur a donc comme rôle d'orchestrer simultanément les lectures

et les écritures sur ces tuyaux. Ceci implique, pour chacun des processus *Acquisition*, de maintenir une table de routage entre numéro de carte bleue et terminaux en attente d'une réponse.

Les échanges entre un serveur d'acquisition et un serveur d'autorisation sont possibles au travers une paire de tuyaux.

En ce qui concerne la connexion entre le serveur interbancaire et les serveurs d'acquisition, la problématique est strictement identique à celle de connexion entre terminaux et serveurs d'acquisition. Il faut utiliser une paire de tuyaux pour connecter chaque serveur d'acquisition au serveur interbancaire. Ceci implique donc également que le processus *Interbancaire* maintient une table de routage entre serveurs d'acquisitions et numéros de cartes bleues.

2.3 Comment tester sans s'y perdre ?

Le schéma proposé ci-dessus comporte un petit défaut tant que les communications entre processus se feront sur la même machine (donc sans utiliser de *socket*, développement proposé en option) : chaque *Terminal* va fonctionner à partir de la même fenêtre (le même terminal, le même *shell*). Tous les affichages vont donc se faire sur cette même fenêtre.

Afin de faire bénéficier chaque processus *Terminal* d'une entrée et d'une sortie standard qui lui soient propres, une astuce peut être utilisée : lors de la création d'un nouveau *Terminal*, recouvrir le processus courant non pas par *Terminal*, mais par `xterm -e Terminal`.

Cette astuce n'est pas très esthétique, et n'est qu'un intermédiaire avant la communication par socket.

2.4 Livrables

Doivent être livrés sous format électronique :

1. Un ensemble de fichier C, amplement commentés, correspondant aux différents programmes constituant la simulation.
2. Un fichier Makefile permettant de compiler tous les programmes (y compris les programmes de test).
3. Un mode d'emploi expliquant comment obtenir une application opérationnelle, comment l'exécuter, et comment la tester.
4. Un manuel technique détaillant l'architecture, le rôle de chaque composant, expliquant comment tester le bon fonctionnement de façon indépendante et justifiant les choix techniques effectués. Ce manuel doit en particulier bien préciser comment le projet répond au cahier des charges (ce qu'il sait faire, ce qu'il ne sait pas faire, et mettre en exergue ses qualités et ses défauts).
5. Dans le manuel technique, on démontrera qu'il ne peut pas y avoir d'interblocage entre les différents processus, pas plus qu'entre les threads d'un même processus.

Tous les documents à produire s'adressent à des ingénieurs généralistes et les rédacteurs doivent donc veiller à donner des explications concises et claires.

Aucun fichier exécutable, fichier objet, fichier de sauvegarde, fichier superflu ou inutile ne devra être transmis avec les livrables.

Une version papier des rapports sera également remise au secrétariat.

3 Conduite du projet

Les étapes qui vous sont suggérés dans cette partie permettent une approche "sereine" de la programmation de ce projet.

3.1 Introduction

3.1.1 Un projet en plusieurs étapes

Le développement du simulateur présenté ici doit être réalisé en plusieurs étapes. Ce découpage conduit le développeur à écrire des programmes indépendants les uns des autres, qui communiqueront entre eux par l'intermédiaire de tuyaux, souvent après redirection des entrées/sorties standards.

3.1.2 Méthode "diviser pour régner"

La constitution de l'application globale par écriture de "petits" programmes indépendants qui communiqueront ensuite entre eux est un gage de réussite : chaque "petit" programme générique peut être écrit, testé et validé indépendamment, et la réalisation du projet devient alors simple et progressive.

La meilleure stratégie à adopter consiste à écrire une version minimale de chaque brique du projet, à faire communiquer ces briques entre elles, puis éventuellement, à enrichir au fur et à mesure chacune des briques. La stratégie consistant à développer à outrance une des briques pour ensuite s'attaquer tardivement au reste du projet conduit généralement au pire des rapport résultat/travail.

Avant toute programmation, conception ou réalisation de ce projet, il est fortement conseillé de lire l'intégralité de l'énoncé, y compris les parties que vous pensez ne pas réaliser, et de s'astreindre à comprendre la structuration en étapes proposée ici.

La traduction de cet énoncé sous forme de schéma est indispensable.

En particulier, il est très important de définir dès le départ l'ensemble des flux de données qui vont transiter d'un processus à un autre, de déterminer comment il est possible de s'assurer que ces flux sont bien envoyés et reçus, puis de programmer les fonctions d'émission et de réception correspondantes. Ces fonctions sont ensuite utilisées dans tous les processus du projet.

Un schéma clair et complet associé à des communications correctement gérées garantissent la réussite de ce projet et simplifient grandement son développement et son débogage.

3.2 Étape 1 : les fonctions de communication

La première étape a été faite pour vous. Elle consiste à formater (selon le protocole définis en annexe) un message complet à partir de ses différents champs et réciproquement à extraire les différents champs à partir d'un message déjà formaté. Les messages ainsi formatés peuvent être lus et écrits dans des fichiers grâce aux librairies programmées en TP, dont une version vous est aussi fournie.

Les problèmes de synchronisation seront abordés dans les étapes suivantes et il s'agit pour l'instant uniquement de traduire sous forme de programmes (de fonctions en C) les protocoles de communication : formattage de messages selon la structure définie en annexe et récupération des informations stockées sous cette forme.

Les fonctions qui ont été faites pour vous sont données dans le dépôt svn initial. Une fois que vous aurez lu le code, et compris son utilisation, vous serez à même de les utiliser ; ces fonctions sont utilisées par tous les processus du projet.

3.3 Étape 2 : réalisation de programmes indépendants

Dans cette deuxième étape, il s'agit de mettre en place les différentes briques du projet et de pouvoir les tester indépendamment.

3.3.1 Processus : *Terminal*

Le processus *Terminal* offre l'interface permettant d'envoyer et de recevoir les informations pour une transaction (montant et numéro de carte). Nous souhaitons utiliser exactement le même code pour tester *Terminal* de manière indépendante et pour l'utiliser dans la simulation globale du système. Pour cela, *Terminal* accepte comme arguments deux descripteurs de fichier vers lesquels il redirige son entrée et sa sortie standard (voir annexe). Ensuite, il lit son entrée standard et écrit sur sa sortie standard. On peut ainsi utiliser *Terminal* :

- en passant 0 et 1 comme arguments, ce qui permet de tester *Terminal* "à la main" en lisant les demandes de paiement à l'écran sur la sortie standard et en lui passant les réponses (autorisation ou non) via l'entrée standard ; ou
- en passant les descripteurs de tuyaux utilisés par *Acquisition*, ce qui permet d'utiliser *Terminal* dans la simulation complète du système.

3.3.2 Processus : *Autorisation*

Le processus *Autorisation* offre l'interface permettant de recevoir des demandes d'autorisation et d'envoyer les réponses à ces demandes. Comme *Terminal*, il accepte comme arguments deux descripteurs de fichier vers lesquels il redirige son entrée et sa sortie standard.

3.3.3 Processus : *Acquisition*

Le processus *Acquisition* reçoit des demandes d'autorisation en provenance soit des terminaux, soit du serveur interbancaire et des réponses en provenance soit du serveur interbancaire soit du serveur d'autorisation. Les ordres seront redirigés ("routés") vers le serveur interbancaire ou bien vers le serveur d'autorisation et les réponses seront redirigés soit vers les terminaux soit vers le serveur interbancaire, conformément au cahier des charges. A cette phase du projet, on peut utiliser :

- des fichiers dans lesquels le processus *Acquisition* écrit lorsqu'il est supposé envoyer un message ;
- des fichiers dans lesquels le processus *Acquisition* lit les messages lorsqu'il s'attend à recevoir ; ces derniers seront préparés "à la main".

Ces fichiers permettent de simuler les échanges avec les processus *Autorisation*, *Terminal* et *Interbancaire*.

Le programme doit accepter sur sa ligne de commande au moins un paramètre représentant la taille de la mémoire servant à la gestion des demandes d'autorisation.

Dans une première phase, le processus *Acquisition* traite *séquentiellement* chaque terminal, puis le serveur d'autorisation et enfin le réseau interbancaire. Dans une seconde phase, on *parallélise* au maximum les requêtes. La première phase doit être pensée et conçue sur le papier afin de pouvoir facilement passer à la seconde phase. La première phase peut suffire pour passer à la suite du projet, mais elle est obligatoire et devra être rendue. On conservera toutes les versions des différentes phases du programme à des fins de test.

3.4 Étape 3 : création d'un réseau bancaire

3.4.1 Préparation de la communication par tuyaux

En pratique, le processus *Acquisition*, le processus *Autorisation* et chaque processus *Terminal* communiquent par une paire unique de tuyaux. Une fois ces deux tuyaux créés, il est possible de modifier le programme *Acquisition* pour qu'il utilise non pas des fichiers mais ces deux tuyaux. Les processus *Terminal* et *Execution* n'ont pas à être modifié.

3.4.2 Raccordement

Pour terminer la mise en place des communications au sein d'une même banque, il faut maintenant créer d'une part une paire de tuyaux entre *Acquisition* et chaque *Terminal* (il y aura autant de paires de tuyaux que de terminaux) et, d'autre part, une paire de tuyaux entre *Acquisition* et *Autorisation*.

Modifier le programme *Acquisition* pour qu'il accepte sur sa ligne de commande les paramètres suivants :

- le nom de la banque à simuler
- le code de 4 chiffres associés à cette banque ;
- le nom du fichier contenant les soldes des comptes clients ;

— le nombre de terminaux à créer.

Créer les tuyaux nécessaires, opérer les clonages et recouvrements nécessaires pour créer les processus *Terminal* et *Autorisation* en nombre suffisant.

3.5 Étape 4 : création du réseau interbancaire

3.5.1 Processus : *Interbancaire*

Chaque serveur d'acquisition est relié au processus *Interbancaire* par une paire de tuyaux. Celle-ci permet à *Interbancaire* de recevoir les messages de demande d'autorisation et de transmettre les réponses en retour, après les avoir routés.

L'architecture à mettre en place entre *Interbancaire* et les différents processus *Acquisition* est similaire à celle mise en place entre chaque processus *Acquisition* et les processus *Terminal* qui y sont reliés.

Dans un premier temps, les messages transitant par *Interbancaire* sont simplement lus et écrits dans des fichiers, sans qu'aucune communication ne soit mise en place.

Dans une première phase, le processus *Interbancaire* traite *séquentiellement* chaque serveur d'acquisition. Dans une deuxième phase, on *parallélise* au maximum les requêtes. La première phase doit être pensée et conçue sur le papier afin de pouvoir facilement passer à la deuxième phase. La première phase peut suffire pour passer à la suite du projet, mais elle est obligatoire et devra être rendue. On conservera toutes les versions des différentes phases du programme à des fins de test.

3.5.2 Raccordement

Pour terminer la mise en place des communications au sein du réseau interbancaire, il faut maintenant créer une paire de tuyaux entre *Interbancaire* et chaque serveur *Acquisition*.

Modifier le programme *Interbancaire* pour qu'il accepte sur sa ligne de commande un fichier de configuration dans lequel on trouve les informations suivantes :

— le nom d'un fichier contenant toutes les banques et les 4 chiffres associés à chaque banque ;
— les noms des fichiers nécessaires au fonctionnement de chaque banque ;
— le nombre de terminaux pour chaque banque.

Modifier *Interbancaire* pour qu'il crée les tuyaux de communication avec les processus *Acquisition* et opère les clonages et recouvrements pour créer les processus *Acquisition* en nombre suffisant.

4 Évolutions complémentaires et optionnelles

4.1 Cumul des transactions

On demande de rendre la simulation plus réaliste en permettant au niveau du serveur d'autorisation de comptabiliser l'ensemble des paiements effectués par une carte, afin de proposer une réponse à la demande d'autorisation qui tienne compte du solde du compte et de l'encours de la carte.

4.2 Délai d'annulation

Dans cette version, les processus Acquisition et Interbancaire doivent pouvoir gérer une fonction d'annulation : une transaction est annulée lorsqu'il s'est écoulé plus de 2 secondes depuis le relais de la demande, sans qu'une réponse ne soit parvenue. Si une réponse parvient ultérieurement, elle sera ignorée.

Ceci suppose que les processus Acquisition et Interbancaire conservent une trace datée de toutes les demandes qu'ils traitent et qu'ils vérifient régulièrement la date de péremption de chaque demande.

Attention : l'annulation d'une demande pour délai trop important ne dispense pas les processus Acquisition et Interbancaire d'envoyer une réponse à l'émetteur de la demande.

La synchronisation (demande, autorisation d'écriture, ordre de lecture) sera mise en oeuvre par l'intermédiaire de l'appel système select().

4.3 Utilisation de `socket`

L'utilisation de communications entre machines ne fait pas partie des objectifs de ce cours. Cependant, afin de rendre le projet plus vivant et plus attractif, nous proposons aux plus débrouillards d'utiliser des `socket` de sorte que la simulation soit plus réaliste :

- chaque terminal communique avec son serveur d'acquisition par des `socket` ;
- chaque serveur d'acquisition communique avec le réseau interbancaire par `socket`.

Les informations nécessaires à l'utilisation des `socket` peuvent facilement se trouver sur internet.

Le travail de mise en œuvre des communications par `socket` doit être entrepris **uniquement après avoir réussi à programmer une application complètement opérationnelle en mode texte**.

Attention : **ne jamais modifier un programme qui fonctionne et toujours travailler sur une copie de celui-ci**.

5 Annexes

5.1 SVN

Subversion est un système permettant de partager un projet (informatique) et d'en gérer les versions successives. Il facilite la synchronisation des différentes versions, éventuellement produites simultanément par plusieurs développeurs. Une version du projet (dépôt) est stocké sur un serveur. Chaque développeur travaille sur une copie locale. Avant chaque session, la copie locale doit être mise à jour et, à l'issue de la session, les modifications apportées sur la copie locale du projet doivent être reportées sur le serveur.

Vous trouverez ci-dessous la liste des commandes svn qui vous seront utiles pour gérer votre projet :

- `svn checkout`
- `svn add/delete`
- `svn commit -m "Message"`
- `svn copy`
- `svn update`
- `svn revert`

Nous vous invitons à consulter le tutoriel svn mis à dispositions par le club nix : http ://www.clubnix.fr/tutoriel_svn

Remarque importante.

- Faire '`svn update`' avant chaque session de travail.
- Faire '`svn commit -m"Message"`' après chaque session de travail.
- Faire '`svn copy trunk/ tags/VersionX.X`' pour garder une trace de chaque version fonctionnelle.

5.2 Codes disponibles sous svn

Un dépôt svn (dont l'adresse vous sera communiqué par ailleurs) sera créé pour vous. Initialement, vous devrez ajouter dans ce ce dépôt certaines fonctions C (décrites Sections 5.4, 5.5 et 5.6) qui vous seront transmises par email et qui pourront vous aider dans la réalisation du projet.

Vous aurez également à votre disposition un générateur aléatoire implémenté dans les fichiers alea.c et alea.h.

Vous trouverez enfin une bibliothèque permettant de lire et écrire une ligne dans un fichier (comme spécifié dans les TP de l'unité 'Système d'exploitation') dans les fichiers LectureEcriture.{c,h}.

5.3 De l'intérêt des standards pour assurer une meilleure interopérabilité

Le recours à des protocoles "standardisés" (ou pour le moins communs) à un double intérêt :

— il permet de gagner du temps sur le développement de ce projet et d'illustrer par la pratique la façon dont les problèmes sont traditionnellement résolus en informatique :
— il permet de mélanger les projets entre eux afin de tester le respect du protocole et peut-être, d'aider certains binômes à avancer (il suffit d'utiliser les processus d'un autre binôme [2]).

La capacité ainsi esquissée à mélanger des composants issus de programmeurs ou prestataires différents pour constituer un système d'information unique se nomme "interopérabilité". C'est un des enjeux majeurs de l'informatique actuelle.

5.4 Protocoles de communication et format de messages

Les messages sont constitués de caractères ASCII (sans accent). Chaque message est constitué de différents champs séparés par les caractères '|' et terminés par un caractère de fin de message '\n'.

Remarque : afin de simplifier au maximum le protocole, on considère qu'il est impossible que plus d'une demande d'autorisation ou réponse, correspondant à une transaction faite avec une carte donnée, ne circule sur le réseau. Cette hypothèse, assez réaliste, permet de simplifier l'appariement des ordres et des accusés de réception au niveau des processus *Acquisition* et *Interbancaire*.

Les échanges utilisent deux types de message, *la demande d'autorisation*, et *la réponse*. Ces deux types de messages partagent un format générique comportant trois champs :

Format : `|C...C|T...T|V...V|\n`

— `C...C` est le numéro de carte bleue, codé sur 16 chiffres, sur laquelle porte le message ;
— `T...T` est le type du message, c'est à dire '`Demande`' ou '`Réponse`' ;
— `V...V` est la valeur associé au message. Dans le cas d'une demande, il s'agit du montant de la transaction exprimé en centimes d'euros. Dans le cas d'une réponses, il s'agit d'un booléen qui vaut '`1`' si la demande est acceptée ou'`0`' si elle est refusée.

Les fichiers message.c et message.h (fournis dans l'archive disponible en ligne) implémentent les fonctions pour créer et interpréter un message formaté selon ce protocole.

5.5 Générateur aléatoire

Le dépôt svn créé pour vous contient également un exemple de générateur aléatoire.

5.6 Redirection des entrées et des sorties

La redirection des entrées et des sorties peut se faire grâce à l'appel système `dup`.

2. Bien entendu, cette utilisation ne peut s'entendre qu'à des fins de mise au point...

```
int dup2(int oldfiledes, int newfiledes);
```

Cet appel système sert à dupliquer le contenu du descripteur `oldfiledes` dans un autre descripteur `newfiledes`. Il renvoie -1 en cas d'échec, et il prend en argument deux descripteurs de fichiers. Par exemple si on a envie que STDIN (==1) soit en fait STDERR (==2) on peut écrire ceci :

```
dup2(2, 1);
```

L'exemple suivant redirige l'entrée et la sortie standard vers un tube, et recouvre le processus courant et son fils par deux programmes. Le père et le fils vont communiquer par leur entrée et sortie standard.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


void usage(char * basename) {
    fprintf(stderr,
        "usage : %s [<programme 1> [<programme 2>]]\n",
        basename);
    exit(1);
}

int main(int argc, char *argv[]) {
    int pid;       /* permet d'identifier qui on est*/
    int fdpipe[2]; /* sera utilisé pour lier les processus */

    if (argc != 3) usage(argv[0]);

    /* on créé le pipe  qui sera utilisé pour relier
       la sortie du premier processus
       vers l'entrée du second
    */
    if ( pipe(fdpipe) == -1 ) {
        perror("pipe");
        exit(-1);
    }

    switch(pid = fork()) {
        case -1:
            /* le fork a échoué */
            perror("fork");
            exit(-1);
        case 0:
            /* code du fils */
            /* on fait en sorte que lorsque le processus
               écrira sur l'entrée standard (1)
               il le fera en fait dans le pipe (fdpipe[1])
            */
```

```
            dup2(fdpipe[1], 1);
            /* on ferme tout, même le pipe...
               on n'en a plus besoin
            */
            close(fdpipe[0]);
            close(fdpipe[1]);
            execlp(argv[1], argv[1],NULL);
            /* pas besoin de break,
               ce code n'existe déjà plus à l'exécution
            */
        default :
            /* code du père */
            /* on fait en sorte que lorsque le processus
               lira sur la sortie standard (0)
               il le fera en fait dans le pipe (fdpipe[0])
            */
            dup2(fdpipe[0], 0);
            close(fdpipe[0]);
            close(fdpipe[1]);
            execlp(argv[2],argv[2], NULL);
    }
   /*
   cette portion de code ne sera jamais exécutée,
   puisque les processus ont déjà
   été remplacé. On met néanmoins un
   return sinon le compilateur proteste.
   */
   return 0;
}
```

Le code ci-dessus permettant de rediriger les entrés et sorties est disponible dans le fichier TestRedirection.c que vous trouverez dans l'archive téléchargées.