Frédéric LETELLIER
Email : frederic@letellier.org
Id CNAM : 04-22882
Année : 2006-2007

Projet méthodes de programmation systèmes 2006-2007

Partie I : Proposer une architecture logicielle pour la partie serveur.

I.1) La communication

Pour réaliser mon serveur, j'ai fait le choix de lui donner 2 ports de communication.

Il y a un port qui sert à recevoir les demandes de consultation de la part des postes clients et un autre port qui sert pour les demandes de virement de la part des postes clients.
Pour ces 2 services, j'utilise des communications par socket avec le protocole TCP.
Si j'ai fait le choix du TCP (SOCK_STREAM), c'est que le flot d'informations échangé est contrôlé. Avec ce protocole, je suis sûr que les données arrivent bien à destination et je suis aussi sûr que cela arrive dans le bon ordre.

Il y aussi une connexion interne qui permet d'échanger des données à travers une file de messages. Pour cette connexion, j'ai défini une CLE.

I.2) Les processus

Il existe des processus lourds et des processus légers.
L'avantage des processus lourds, c'est que c'est plus sécurisant compte tenu que les données sont indépendantes dans chaque processus. Un défaut des processus lourds sont que cela prend plus de ressources sur la machine. Un problème du fork, c'est qu'il n'est pas très portable d'un environnement à un autre. Cela signifie qu'un programme écrit pour Linux/UNIX risque de ne pas fonctionner sous Windows car Windows utilise d'autres méthodes pour créer des processus.

Quand cela est possible (c'est-à-dire quand cela ne provoque pas de problème), il est préférable d'utiliser des processus légers (thread). Dans le cas de ce projet, j'ai considéré que cela ne posait pas de problème particulier d'utiliser des processus légers et c'est donc ce que j'ai choisi.

Partie II : Primitives de communication et appels systèmes

II.1) Organisation du projet

Dans mon archive projet, vous devriez trouver les documents suivants :

- rapport.pdf

- dossier /pseudo code
- /pseudo code /client_pseudo_code.pdf
- /pseudo code /serveur_pseudo_code.pdf
- /pseudo code /banques_pseudo_code.pdf

- dossier /code source
- /code source /client.pdf
- /code source /serveur.pdf
- /code source /banques.pdf

Avant d'écrire un programme, je commence toujours par écrire les fichiers .h qui définissent les fonctions, puis j'écris dans chaque fonction ce qui doit être fait. J'écris cela dans une sorte de pseudo-code de haut niveau (en langage naturel).
C'est seulement une fois que j'ai fini de définir l'architecture de mon projet que je commence à coder.
Dans le dossier « pseudo code », vous trouverez le squelette du projet.
J'espère que cela correspond à ce que vous attendez. Si ce n'était pas le cas, n'hésitez pas à me le dire et je ferai de mon mieux pour vous envoyer dans la journée une nouvelle version qui tiendra compte de vos remarques.

Dans le dossier « code source » se trouve le projet complet codé en C++.

J'ai décomposé le projet en 3 parties.
Il y a une partie « banques » qui permet aux autres banques de communiquer avec le serveur à l'aide d'une file de messages.
Il y a une partie « client » qui permet de se connecter sur le serveur et de lui demander soit de consulter son solde, soit d'opérer un virement.
Enfin, il y a une partie « serveur » qui communique avec les clients et qui de façon périodique exécute les demandes faites à travers la file de messages.

II.2) Les primitives de communication et appels système

Une socket est une interface de communication réseau. Il s'agit d'un point d'accès aux services de la couche transport, c'est-à-dire TCP ou UDP. La communication par sockets sur un réseau adopte généralement un modèle client-serveur ; en d'autres termes pour communiquer il faut créer un serveur prêt à recevoir les requêtes d'un client.

Dans tous les cas, avant d'utiliser une socket il faut la créer, c'est-à-dire créer un descripteur associé à l'ensemble des informations constituant la socket (buffers, adresse, port, état, etc.). Ensuite il est éventuellement possible de l'attacher à une adresse représentant la provenance des messages envoyés.

Côté serveur, la création de la socket est suivie d'une mise en attente de message dans le cas d'une communication UDP, ou de mise en attente de connexion dans le cas d'une communication TCP. Dans le cas d'une communication TCP, il est généralement profitable de

permettre au serveur de gérer plusieurs connexions simultanées ; dans ce cas un nouveau processus sera créé pour chaque connexion ou bien les sockets seront enregistrées pour être utilisées dans un appel système select(2).

Côté client, la communication se fait tout d'abord en renseignant l'adresse du serveur à contacter. Ensuite peut avoir lieu l'envoi proprement dit de données ou la demande de connexion (selon le cas).

II.2.a) Le processus client

Voici l'algorithme du client que j'ai utilisé pour me connecter sur le serveur :
- création de la socket,
- construction de l'adresse du serveur,
- demande de la connexion
- dialogue avec le serveur.

La demande de connexion se fait grâce à :
int connect(sock,p_adr,lgadr)
la valeur de retour 0 indique qu'il y a réussite et que la socket du client est connectée à une socket correspondant à la connexion pendante chez le serveur.

L'émission se fait par la primitive
int send(sock,msg,lg,option).
La variable option peut prendre deux valeurs MSG_OOB dans le cas de message urgent, 0 sinon. Ce dernier cas est équivalent à la primitive write.

La réception se fait par la primitive
int recv(sock,msg,lg,option).
La variable option peut prendre trois valeurs MSG_OOB dans le cas de message urgent, MSG_PEEK dans le cas où on consulte le message sans le consommer, 0 sinon. Ce dernier cas est équivalent à la primitive read.

L'arrêt de communication se fait par la primitive
int shutdown(sock,sens).
Elle permet de signaler que sur la socket connectée par le descripteur sock elle ne veut plus recevoir (sens=0), émettre (sens=1), recevoir ou émettre (sens=2).

II.2.b) Le processus serveur

Voici l'algorithme du serveur que j'ai utilisé pour communiquer avec les clients :
L'algorithme du serveur est :
- création et attachement de la socket d'écoute,
- ouverture du service par écoute
- boucle infinie consistant à attendre une demande et créer un sous processus de traitement.

La primitive de création de file d'attente de connexions entrantes
int listen (sock,nb)

La primitive d'acceptation de connexion qui renvoie un descripteur de socket
int accept(sock,p_adr,p_lgadr)

Quelques remarques

Je pense que le programme fait ce qui était demandé mais j'ai un doute sur la partie virement.

Quand je lis l'énoncé concernant le client, j'ai l'impression que le client envoie l'ID, le nom, le prénom et le mot de passe du compte A, puis l'ID du compte B et enfin, la somme à virer. Quand je lis l'énoncé de la partie serveur, il est dit que le client doit s'identifier sur les comptes A et B, et donc, j'en déduis qu'il faut aussi envoyer le nom, le prénom et le mot de passe du compte B.

Finalement, j'ai décidé dans le serveur avait besoin de connaître le nom, le prénom et le mot de passe des comptes A et B.
Si j'ai mal compris ce qu'il fallait faire, je peux en quelques minutes rectifier le programme pour qu'il fasse bien ce qu'on attend de lui.