RAPPORT DE RÉSEAUX

[par Mike Duran & Damien Ledda]

Préface

Ce document a pour but de décrire le déroulement de notre projet de Réseaux. C'est le résultat du travail qui nous a permis de réaliser un réseau client/serveur (accès)/serveur (données). Ce rapport contient l'ensemble des éléments du projet, d'un point de vue technique et d'un point de vue pédagogique. Nous vous expliquerons comment nous avons imaginé le concept, puis en présentant la communication des messages des programmes. Pour conclure ce rapport, nous vous présenterons la manière dont nous avons géré le projet, puis nous ferons le point sur les différentes difficultés rencontrées.

Plan technique du projet

La structure de notre projet est inspiré du schéma présenté dans l'énoncé. les informations des clients (username, mots de passe et champs accessibles) sont stockées dans un fichier appelé `fichier.txt`, sous la forme `identifiant_client:mot_de_passe:champ_acces`. Le champ_acces est un champ optionnel. Il peut bien sur contenir plusieurs champs accessibles. Tout notre programme va se baser sur ce fichier (mais pas que). Notre projet est assisté de deux autres fichiers: `taille.txt` et `age.txt`. Ces deux fichiers sont les informations que les serveurs "spécialisés" vont prendre. Les informations sont organisées de la facon suivante: `identifiant_client;valeur_taille` pour le fichier `taille.txt`, `identifiant_client;valeur_age` pour le fichier `age.txt`.

Client

La manière de utiliser le programme clientUDP est la suivante: `./clientUDP '0.0.0.0' 2066`, le premier argument étant la l'adresse IP et le deuxième argument, le port du serveur. Dans notre cas le port du serveur d'accès est *2066*. C'est ce qui permet au client de communiquer avec le serveur d'accès. Après le lancement :

- le client donne le login
- Appuie sur ENTER
- donne le mot de passe
- Appuie sur ENTER

Les valeurs sont recherchés dans le fichier ouvert par le serveur d'accès et le client s'attend à deux réponses possibles:

- soit l'affichage de bienvenue confirmant sa connection au serveur d'accès.
- soit `YOU SHALL NOT PASS!!!!` (oui, on a choisi de faire une référence culturelle.) indiquant le refus de connection au seveur d'accès (il faudra donc relancer le programme).

Dans le cas d'une connection reussite une ligne s'affiche: `Your request is...`. L'utilisateur a le choix de plusieurs options, représentant les possibilités de communication du client au(x) serveur(s):

- `lire`: l'utilisateur donne un ou plusieurs champ(s) qu'il veut vérifier. Si il a accès au champ, il recoit la valeur sauvegardée dans un serveur correspondant à sa requête, sinon il recoit le nom du champ auquel il n'a pas accès.
- `ecrire`: Le couple `champ_a_modifier:valeur` est requis, et si le client a accès au champ alors la valeur du champ sera modifiée dans le serveur spécifique, et il recoit une confirmation de cette modification. `écriture modifié`. Sinon le champ auquel il n'avait pas accès sera renvoyé sur l'écran du client.
- `supprimer`: Aucun paramètre à donner, il met les valeurs du client dans les deux serveurs de stockage de données à *-1* (considéré comme *N/A*).
- `help`: Aucun paramètre à donner, il affiche un mini tutoriel comment utiliser le programme du client avec chaque requête. Oui, on a voulu guider un peu le client, et ce n'était pas au programme.
- `bye`: Aucun paramètre à donner, les valeurs du client sont enregistrées dans les fichiers spécifiques par les serveurs de données et il termine le programme en étaignant les serveurs et sortie du programme client.

Serveur d'accès

Le serveur d'accès est le serveur qui lie le client et le serveur de stockage de données. La manière de le déclencher est de lancer dans un autre terminal `./serveurUDP`. Il est impératif de le lancer avant de tenter la connexion du client sinon il attendra une réponse infiniment. Il fait les binding nécessaires pour les sockets, puis il se met en attente du login et du mot de passe du client. Dans le cas d'une erreur il renvoie un message d'erreur au client, et attend que celui ci puisse se (re)connecter. Dans le cas d'une reussite, le programme entre dans une boucle infinie qui attend les requêtes du client, après envoi de la confirmation `Logged in`. À chaque fois qu'une requête est prise en compte, il envoie "done" au client afin que le client puisse en envoyer une nouvelle et se remet en attente.

- Si réception de `bye`, il envoie au deux serveurs de stockages de données le signal de terminaison afin qu'ils puissent respectivement éxécuter leur tâche qu'est celle de sauvegarder les données dans les fichiers. Puis il envoie "Done" au client comme confirmation de fin de requête avant de se terminer.
- Si réception de `lire`, il prend chaque champ, teste s'il est contenu dans la ligne où le login du client est situé. S'il est dedans, il est envoyé au serveur de données spécifique et attend la valeur renvoyée par celui-ci avant de le renvoyer au client. Dans le cas inverse, il envoi au client le nom du champ auquel il n'a pas accès.
- Si réception de `ecrire`, le déroulement reste similaire à `lire`, mais avec transmission d'une valeur à modifier et la nouvelle valeur. Si réception de la confirmation d'écriture, la variable est enregistrée dans le derveur de données.

- Si réception de `supprimer`, on envoie la requête au deux serveurs afin qu'elles écrivent *-1* en tant que valeur *N/A*. On considère ce choix comme étant judicieux car il est fréquent dans le travail avec les primitives systèmes dont les valeurs d'érreur sont *-1*. (On voulait respecter cette tradition.) Le serveur d'accès renvoie ainsi la confirmation de suppression des champs au client.
- Si réception de `help`, des instructions de fonctionnement de ce que le client est capable de faire seront renvoyées au client.
- Sinon, le serveur indique que c'est une requete inconnue et le client doit en renvoyer une nouvelle par un formulaire d'utilisation des requêtes.

Serveur de stockage des données

Le serveur de stockage de données est le serveur qui selectionne les données des fichiers 'taille.txt' et 'age.txt', ses executions sur le terminal sont : `./serveurData taille.txt 2067` et `./serveurData age.txt 2069`, le port *2067* étant pour la taille et *2069* pour l'âge. Nous pouvons (et devons) déclencher le programme deux fois pour utiliser les deux fichiers. Lors de son déclenchement, il enregistre dans une table à deux dimensions les données lues dans le fichier donné en argument, et attend une requête de la part du serveur (donc du client).

- Si réception de `bye`, il enregistre les données stockées dans le fichier donné en argument, et se termine.
- Si réception de `lire`, il vérifie si le client est présent dans sa liste, et renvoie la donnée au serveur d'accès.
- Si réception de `ecrire`, il vérifie (aussi) si le client est présent dans sa liste, et écrase l'ancienne valeur par celle reçue depuis le serveur d'accès et envoie une confirmation d'écriture.
- Si réception de `supprimer`, le déroulement est similaire à `ecrire`, mais avec la valeur *-1*.

Messages

Nous allons donc expliquer les différents méssages que les programme aura à confronter respectivement.

`ClientUDP`

Lors du login réussi: `Your request is...` sera affiché sur le terminal du client. La raison de cette écriture est que l'utilisateur puisse savoir qu'il peut envoyer les messages à tout moment. Ainsi viennent les requetes.

- `lire champ champ...`: il recoit la valeur du champ respectif s'il a accès à ce dernier. Dans le cas contraire, il obtiendra le nom du champ auquel il n'a pas accès.
- `ecrire champ:val champ:val...` ou `supprimer`: le principe reste exactement le meme que lire juste qu'au lieu de la valeur du champ, en cas de reussite, il reçoit un message de confirmation de la reussite de la modification: `ecriture modifiée`.
- `bye`: les messages de déconnexion seront affichées afin de confirmer étape par étape au client que sa déconnexion se déroule bien et qu'il n'a qu'à attendre la déconnexion.

`ServeurUDP`

Quand un client se connecte sur le serveur, sur l'écran du serveur sera affiché: `j'attend...` pour que l'utilisateur sache que le serveur d'accès est prêt à recevoir une requete. Si une requête est reçue, il affiche la requette prise en compte par le client par `Request 'requte_du_client' received`. Dépendant de la

requête, il affiche si les executions sont confirmées. À la fin de la requete il envoi `done` pour confirmer que le client peut envoyer une nouvelle requête. Il affiche `Done proceeding request 'requete_du_client'`.

`ServeurData`

Quand le serveur est lancé, il affiche `Waiting for a request from a client...` suivi d'un affichage de tous les noms qui sont dans le fichier respectif (`Reading data from nom_client...`). Puis il arrive en mode attente: `j'attend...`. Quand il reçoit une requête, il affiche `'request received: 'request_du_client'` pour nous confirmer qu'il a reçu la requête sans le moindre problème, suivi du message de début de fonction `'requete_du_client' initiated`, suivi de l'envoi de la confirmation de la requete au serveur d'accès. Quand `bye` est reçu, le programme se termine en confirmant l'écriture des nouvelles valeurs dans le fichier prise en compte: `Writing data to nom_client...` avant de se terminer.

La répartition du projet

Au cours de ce projet nous nous sommes pas répartis des taches précisément, nous avons travaillé étape par étape. Nous avons ainsi décidé qu'avant tout, on arrivais à faire en sorte que le schéma du projet fonctionnait sur un seul client, ensuite d'adapter le corps principal de notre programme sur plusieurs clients (même si on ne peut que se connecter qu'avec un seul par serveur d'accès).

Le programme qu'on vous présente ici est ainsi le corps principal. Les conditions du projet sont ainsi fonctionnelles sur un seul client. Par rapport aux autres tâches, nous avons pris la décision de construire les bases principales du projet. Les principales difficultés rencontrées étaient surtout sur la structuration du programme, et gérer le transfert des chaines de caractères, car quand on envoyait nos chaînes de caractères, des caractères inconnus ont été rajoutés à la chaine transférée. Nous avons corrigé cela en n'utilisant moins de malloc. Nous avons ainsi utilisé des variables locales dans la majorité des cas.

Il existait aussi que nos fonctions ne fonctionnaient pas, car pour attraper les champs des requetes, nous avons utilisé la fonction `strtok()`. Cette fonction n'était pas thread-safe. Nous avons donc utiliser `strtok_r` qui simplifiait la tâche. On eut ainsi l'idée de travailler avec un tableau à deux dimensions, pour les données dans serveurData afin de simplifier la gestion de ces derniers.

Conclusion

Pour conclure, ce projet était très enrichissant, il nous a permis d'améliorer nos performances en programmation C ainsi sur la compréhension du cours de Réseaux. Il nous était difficile de gérer notre temps pour finir ce projet, dû à d'autres incrustés dans la même timeline. Cela dit, les compétences que nous avons obtenues pour ce projet sont présentes surtout sur le coté pédagogique. Le travail d'équipe était présent, ainsi qu'un partage d'idée très enrichissant.