Projet de programmation système : Dazibao

Réalisé par : Sebastien Ciupek, Clément Charasson, Alain Dias

Introduction

Ce rapport à pour objectif de vous décrire sommairement la démarche qui a conduit à l'implémentation d'aujourd'hui.

Dans ce rapport, nous verrons dans un premier temps une présentation des différentes structures utilisées dans notre code. C'est ensuite que nous reviendrons sur les parties principales de notre code, tels que la lecture ainsi que l'écriture. Finalement nous traiterons de notre choix de conception des notifications.

1. Organisation et conception

Il nous a fallu réfléchir à la conception ainsi qu'à la répartition des tâches. C'est durant cette étape que nous avons pris la décision de créer différentes structures qui seront réutilisées par tout notre code afin de le rendre plus clair et évolutif.

Cette étape de conception nous a permis de distinguer deux grandes parties concernant la manipulation d'un Dazibao. En effet nous avons d'une part la lecture de ce fichier, ainsi que la création et modification du fichier ; c'est à dire l'écriture. Mais il nous a fallu également réfléchir à l'intégration d'un système de notifications avec une architecture Client/Serveur.

Pour garder une cohérence dans le code du projet quelques règles d'implémentations furent nécéssaire avant de coder :
1-faire des commentaires pour les différentes fonctions
2-indenter le code
3-éviter le plus possible de faire des malloc/realloc et donc tout faire en statique
4-faire en sorte, si possible, de ne pas faire de la duplication de code
5-avoir des noms de fonctions et de variables explicites
6-écrire dans `stderr` les erreurs (utiliser `perror()`) en utilisant le formalisme suivant : <"type d'erreur" : "fonction où l'on ce trouve">. Cela permet de retrouver l'endroit de l'erreur plus facilement.

2. Structures principales de notre programme (`dazibao.h` et `tlv.h`)

Dans un premier temps, nous avons voulu manipuler chaque `tlv` de manière individuelle, cela nous a donc poussé à créer une structure `tlv` avec ses champs respectif type, longueur, valeur. Le champ `valeur` est représentée par une position dans le fichier dazibao. Cette position correspond au début du fichier qui sera contenu à l'interieur du `tlv`. La structure `tlv` possède également un champ `nb_sous_tlv` pour les compounds et les dated.

En effet, une implémentation avec cette structure nous permet de centraliser toutes les informations d'un `tlv` lu.

Ensuite, afin de garder une trace de chaque `tlv` lu, nous avons pris la decision d'enregistrer ces `tlv` dans un tableau (`tab_tlv)`, ainsi nous pouvons donc nous référer a ce tableau pour afficher tous les `tlv` du dazibao. Mais aussi, grâce à ce tableau, nous pouvons supprimer un `tlv` en particulier ainsi qu'effectuer la compaction du dazibao de manière simple.

Nous avons ensuite, crée une structure dazibao contenant ce même tableau de `tlv`. Ceci afin de garder un lien entre le chemin du dazibao en tant que fichier et le contenu de ce même fichier dans le tableau de `tlv`. Ces deux structures constituent la base de notre implementation du programme.

Une dernière structure fut créé, c'est la structure de notifications (`notif`). Elle nous permet de connaitre la liste des dazibao suivis par l'utilisateur, mais aussi l'état de la connexion avec le serveur de notifications. Elle contient la socket permettant de dialoguer avec le serveur. Cette structure de notifications est quant à elle la base de notre implémentation des notifications et nous y reviendrons plus tard dans ce rapport.

3. Lecture et affichage du Dazibao (fichiers `dazibao.c` et `tlv.c`)

La lecture complete du fichier `dazibao` va entrainer le remplissage du tableau de structure `tlv` contenu dans la structure `dazibao`. Pour se faire il faut gérer l'ouverture du fichier et s'assurer que le dazibao est valide. De plus, nous devons verrouiller le fichier pendant toute la durée de l'operation de lecture, c'est ainsi que nous avons décidé de séparer cette opération en deux temps.

Dans un premier temps, nous ouvrons le fichier en lecture seule (dazibao valide), puis nous plaçons un verrou de type partagé.Ensuite, nous commençons la lecture une a une de chaque `tlv`. La fonction de lecture écrit le contenu du `tlv` dans le tableau `tab_tlv`.
Finalement, lorsque la lecture est complète, nous déverrouillons le fichier dazibao et le fermons.

Lorsque la lecture est terminée, il faut être capable d'afficher le contenu précédemment lu. C'est le role de la fonction `listeTypesDazibao` qui est un affichage basique. Elle permet d'afficher les types des `tlv` rencontrés dans le fichier. Mais cet affichage étant assez pauvre, nous avons donc décidé de coder une fonction d'extraction de `tlv` afin de récupérer un fichier se trouvant dans le Dazibao.

4. Ecriture dans un dazibao : (insertion.c)

L'idée de l'écriture (fichier insertion.c) est de réunir toutes les fonctions qui vont permettre d'écrire dans un fichier dazibao. Comme leurs nom l'indique les tlv sont composés d'un type, d'une longueur et d'une valeur. C'est pourquoi nous avons crée des fonctions de base pour écrire dans un fichier dazibao, nommés insert_type, insert_longeur, insert_valeur avec les bon paramètres. Cela permet de réduir la duplication de code et donc d'avoir un code plus solide. Ainsi si je veux créer l'insertion d'un tlv simple comme par exemple un tlv texte (qui se nomme insert_txt) j'utilise ces fonctions.

Du point de vue externe nous n'utilisons qu'une seule fonction nommée insert_tlv qui interragie avec l'utilisateur et fait appel aux fonctions "insert" correspondantes.

Pour les tlv plus complexes comme par exemple le tlv dated ou compound nous faisons un appel récurssif à la fonction insert_tlv. Car nous ne savons pas combien de sous tlv sont créé à l'interieur d'un compound.

5. Etape suivante : la réunion du code (interface, écriture.c et lecture.c)

La deuxième grande étape de notre developpement fut de réunir les codes existants autour d'une interface.

Nous nous sommes donc imposés quelques règles pour l'interface. Elle doit être la seule à interagir avec l'utilisateur. Elle doit appeler des fonctions simples avec un minimum de paramètres et si possible toujours les mêmes. Et avoir la possibilité d'ajouter de nouvelles fonctionnalités (suppression ,compaction…) en respectant les règles précedement citées.

Plusieurs problèmes ont été soulevés à ce moment là. L'un des principaux fut que les paramètres de fonctions étaient très différents entre les fonctions d'écritures (fichier insertion.c) et de lecture (fichier dazibao.c et tlv.c). Cela rend plus compliqué la compréhention et la cohérence du code. Le code étant très avancé il n'était pas possible de révenir en arrière.

L'idée fut donc de faire des fonctions intermédières entre l'interface et les fonctions existantes. Elles seront plus simple à utiliser et cacherons les différents appels aux fonctions qui ont déjà été implémentées.

Concretement ces fonctions intermédiaires se trouvent dans les fichiers ecriture.c et lecture.c. Elles prennent toutes en paramètre une structure dazibao.

Cette solution permet de répondre à un autre gros problème c'est de pouvoir intégrer de nouvelles fonctionnalités plus facilement dans le code principale sans géner le travail des autres. Les fonctions qui n'étaient pas encore été codées étaient laissé vides. Cela nous a permis de

travailler en parallèle chacun sur une fonction. Une fois le code fait il était récupéré par les autres membres du groupe. C'est ainsi que les codes de la suppression, de la compaction et l'extraction ont pu êtres développés en paralèlles.

6. Notification d'un dazibao : (fichier `serveur.c` et `client.c`)

Le dernier apport au code fut l'ajout de la notification. La notification permet de savoir si un dazibao a été modifié par un programme. Cette vision correspond à une architecture client / serveur.

Le serveur :

Dans notre code le serveur doit être exécuté manuellement. Si un client modifie un dazibao il se connecte au serveur via une socket et envoie un message correspondant au dazibao modifié. Pour gérer plusieurs client à la fois et créer une attente passive le serveur génère un thread (nommé `attendre_client`) qui écoute chaques clients. Et chaque client est enregistré dans un tableau (`tab_client`). Un thread (nommé `notification_client`) sera chargé d'envoyer le message à la liste des clients connectés. Si un client se ferme un signal EPIPE est intercepté et stop le thread correspondant.

Le client :

Le client c'est notre programme, il va se connecter au serveur et éxcuter un thread d'écoute (`thread_notifications_main`). Si le serveur n'est pas exécuté le thread n'est exécuté non plus. Une fonction nommée `client_ecriture` permet d'envoyer un message sur la socket pour dire si nous avons effectué une modification dans le dazibao. Toutes les notifications sont envoyé aux différents clients. La structure `notif` contient un tableau (`dazibao_a_suivre`) qui nous permet de filtrer les message reçu. En revanche, si le client n'est pas connecté au moment d'une modification du dazibao, alors il essaye de se connecter automatiquement au serveur.

7. Extensions au projet

Nous avons à certains moments de notre implémentation du projet décidé d'ajouter quelques fonctionnalitées.

Concernant l'ajout de nouveaux TLV au dazibao, nous avons voulu bloquer des entrées incorrectes. Nous empechons l'ajout de fichiers texte ne respectant pas le format UTF-8. L'ajout enpèche aussi tout ce qui produit un TLV incohérent. En effet, il ne doit pas être possible d'ajouter un fichier texte en le faisant passer pour une image JPEG.

Grace au tableau de tlv nous avons pu realiser assez facilement la suppression d'un tlv. Ce tableau nous permet aussi de faire une compaction complète du fichier. C'est à dire retirer les pad1 et padN contenu à l'interieur des TLV de type `compound` ou bien de type `dated`. Ainsi, nous pouvons obtenir le maximum de gain d'espace disque apres une compaction.

8. Amélioration possibles du code

Notre implémentation du projet nous permet un ajout de fonctionnalitées supplémentaires de manière assez aisée.

Notre application étant basée principalement sur un tableau de TLV de type statique, nous pourrions le rendre dynamique, sous forme de liste. Ce changement apporterai un gain de memoire dans le cas de dazibao très petits.

De plus, lors d'un ajout ou d'une suppression de TLV au dazibao, si notre programme reçoit une interruption, ou bien rencontre une erreur, il y a un fort risque de corruption du fichier dazibao. Il faudrait dans ce cas, implémenter une gestion plus poussée des interruptions afin de pouvoir tronquer l'ajout s'il n'a pas été effectué jusqu'au bout.

Une interface graphique pourrait également être implémenter, afin de rendre l'utilisation de l'application plus intuitive et plaisante. On pourrait ainsi, afficher directement le contenu de TLV à l'écran et ne plus devoir passer forcément par une extraction.