Compte rendu de projet de Programmation Système

Adeline Hirsch
Cynthia Legrand
Chloé Dequeker (Colin)

Sommaire :

I - Introduction

Le but de ce projet était de réaliser un « dazibao ». Un dazibao est un fichier contenant une suite de messages de différents types (texte, image – png ou jpeg – , dated, compound), et possédant une entête particulière stockée sur 4 octets, permettant de distinguer un fichier dazibao d'un autre fichier.

a) Format et TLV

Un fichier dazibao est formaté d'une certaine façon, afin de pouvoir le lire et de savoir quels sont les types des messages, ainsi que leur longueur.
Un message (TLV) possède principalement trois champs. Le premier champ est son type, qui doit être un nombre entier comprit entre 0 et 255. Le deuxième champ est sa longueur, qui est stockée sur trois octets. La taille d'un TLV est donc limitée par la valeur max d'un nombre entier stocké sur 3 octets (cette valeur équivaut à 16777216 : On peut donc avoir un TLV d'une taille de ~16,7 Mo). Le dernier champ d'un TLV est son contenu : C'est là que les informations relatives au message vont être stockées (contenu d'une image dans le cas d'un TLV-image, char* dans le cas d'un TLV-texte, etc...).

Il est à noter que les TLV-dated possèdent un champ particulier après leur champ longueur : La date, en secondes depuis 1970 stockée sur 4 octets. Ce qui leur fait office de contenu est un TLV de type arbitraire.
De même pour les TLV-compound, ce qui fait office de contenu est un nombre arbitraire de TLV-s de type-s également arbitraire-s.

b) But du projet

Le but du projet était de mettre au point un programme permettant d'ajouter des TLV à un fichier dazibao, en choisissant leur type et leur contenu, de supprimer des TLV déjà contenus dans le fichier, de lire le fichier et d'afficher son contenu, et de réaliser une compression.
La compression permettra de supprimer certains TLV cachés (connus sous Pad 1 ou Pad N) afin de réduire la taille du fichier.

II – Présentation de l'interface

L'interface qui a été utilisée est gtk 2.0+, qui utilise l'interface gnome. Son utilisation permet une interface graphique intuitive. Un des problèmes rencontrés avec cette interface est la présence d'un nombre important de fuites mémoires.

III – Variables globales

Un certain nombre de variables globales ont été utilisées. Ces variables ont été utilisées pour des données qui étaient nécessaires à un grand nombre de fonctions (ex : L'existence d'une fenêtre gtk, le nombre de TLV-s présent-s dans le fichier, etc.).

Notamment, en plus du nombre de TLVs dans le fichier sont notées dans un tableau les positions de chaque TLV du fichier. Cela va permettre de pouvoir accéder à un TLV en particulier en se plaçant directement dans le fichier, où en accédant directement à la x-ème position du le fichier (grâce à mmap par exemple).

IV - Suppression

La fonction de suppression a été réalisée grâce à l'utilisation de mmap, memset et d'autres fonctions telles que truncate, flock, etc.
Si un TLV à supprimer se trouve être le dernier TLV du fichier, alors le fichier est simplement tronqué. Sinon, les valeurs du TLV à supprimer (type, lenght) sont modifiées/lues en utilisant mmap. Connaissant la position du TLV dans le fichier, il est beaucoup plus simple de simplement lire une valeur plutôt que d'avoir à se déplacer dans le fichier pour pouvoir modifier ou lire celle-ci.
Une fois que la taille du TLV a été lue, son contenu est initialisé à 0 grâce à memset.

V – Compression

La compression se fait également aisément une fois que l'on possède les positions de nos TLV dans le fichier. Le principe est de stocker un TLV en le lisant puis stockant son contenu (ainsi que son type et sa taille), puis de réécrire tout le fichier en partant du début. Cela aura pour effet de réécrire par dessus tous les Pad 1 et Pad N dont les positions sont ignorées. Une fois que tous les messages ont été réécris les uns à la suite des autres, le fichier est tronqué à sa nouvelle taille. Il n'y a donc plus de possibilité qu'un Pad 1 ou Pad N soit présent.
Le défaut de cette méthode serait de l'effectuer alors qu'il n'y a aucune compression à faire : Le fichier va être entièrement réécrit alors qu'il était déjà compressé.

VI – Lecture

La lecture est faite par une première fonction A qui va appeler la fonction B de lecture d'un TLV. La fonction B va retourner la valeur lenght du TLV lu à la fonction A. Tant que la somme des tailles de TLV reçues par A n'est pas supérieure ou égale à la taille du fichier dazibao cela signifie que d'autres TLVs restent à lire, la fonction continue donc.
La fonction B va utiliser la fonction l'appel système read pour lire le contenu du fichier dazibao.
A chaque appel de la fonction B, une vérification est faite sur des variables globales, permettant de savoir si la lecture est faite sur un TLV contenu dans un autre, ce qui reviendra à ne pas tenir compte de sa position, et de ne pas incrémenter le nombre de messages contenus dans le fichier dazibao.

On notera que toutes les lectures ou écritures sont faites soit grâce aux appels système « read » et « write », ou alors avec mmap.

VII – Notifications

Les notifications se séparent en deux programmes distincts : le serveur et le client.
Le serveur va utiliser les sockets unix et va pouvoir surveiller les dazibao-s donné-s en paramètre.
Le serveur peut prendre deux types de paramètres. Soit un fichier contenant le path vers un dazibao (relatif ou absolu), à raison de un chemin par ligne, ou alors un nombre arbitraire de chemin vers des dazibaos en argument.

Le moyen de notification est resté au niveau d'un simple message dans le terminal, mais a l'avantage de tenir au courant l'utilisateur même lorsque l'interface graphique n'est pas disponible (démarrage du système en mode console, ou simplement arrêt de l'interface graphique), il est toujours bon d'être au courant des dernières modifications.

a) Comparaison fork/pthread

L'utilisation de fork plutôt que pthread pour gérer l'ensemble des clients vient du fait que la liste des dazibaos est une variable globale dans le fichier source.
Dans le cas de l'utilisation de pthread, lorsqu'un des thread va remarquer qu'un des dazibaos est modifié (grâce à un flag), il va en informer le client et réinitialiser ce flag. Dans pthread, les variables entre les différents threads sont partagées, et le thread suivant s'occupant d'un autre client ne va pas informer celui ci car le flag aura déjà été modifié par le premier thread. Il s'agit donc de problèmes de concurrences entre les threads.

L'utilisation des fork permet de séparer les données puisqu'il existe le mécanisme du « copy on write » qui indique que les valeurs du processus child sont séparées de celles du processus parent à partir du moment où il y a modification.
Chacun des processus child peut donc modifier indépendamment des autres ses valeurs.

b) Client

Le client quant à lui se lance sans paramètre, se contentant de se connecter à la socket ~/.dazibao-notification-socket et d'attendre un message du serveur. Lorsque le message du serveur est reçu, il est affiché sur la sortie standard s'il s'agit d'un changement connu (message commençant par 'C'), et un message indiquant que le message n'est pas connu est affiché sinon.