Dazibao

Projet de Nicolas Cailloux, Jakub Kaluza et Pawel Mlynarski.

##Gestionnaire de dazibaos

Un dazibao est comme un mur sur votre réseau social favori. Dans ce projet de Master, il peut contenir certains types de données (TLV):
* textes (Text)
* images (JPEG / PNG)
* données "vides" de taille 1 ou N (Pad1 / PadN)
* ensemble de données (Compound)

Chacun de ces types peut être daté (Dated) ou non.
Nous avons ajouté des nouveaux types (voir la partie Écriture)

On dispose de plusieurs fonctionnalités :
* lecture d'un dazibao
* ajout d'éléments à un dazibao
* suppression d'un élément
* compression
* serveur de notifications

La version actuelle du programme est la 0.53 : tout fichier dont l'entête ne correspond pas à 53 0 sera rejeté.

Les informations sur la compilation et l'utilisation du programme se trouvent dans le fichier README.

Lecture

La lecture se fait via le programme `reader`.

`reader <options> file`

Par défaut, le fichier spécifié est affiché en brut sur la sortie standard (les textes sont traités, mais les images et les vidéos ne sont pas affichées). Pour visionner tous les éléments, notre programme crée un fichier html avec l'option -h . Pour l'affichage, d'autres possibilités envisagées étaient un programme SDL ou une interface GTK mais nous avons pris le choix du html pour garantir la portabilité de notre programme.

Implémentation

Pour la lecture de dazibao, nous disposons de "reader.c" et "reader.h". Le processus d'éxécution est le suivant :
- vérification du format de la commande (présence d'un fichier en argument)
- appel à `readdazibao(fichier)`, qui va
  - ouvrir le fichier en lecture et mettre un verrou flock partagé dessus

grâce à `lockopen`
  - vérifier l'entête (numéro magic et de version) via `checkversion`
  -si on est en mode HTML, on va imprimer lenrobage, avec `printHTML`
  - récupérer sa taille, et chercher des TLV (`loop`) entre l'indice 4 et l'indice correspondant à sa taille :
    - la boucle se fait entre deux indices du fichier, et on chercher des TLV grâce à `readTLV`
    - on commence par allouer de la mémoire pour un TLV
    - on lit d'abord 1 octet : c'est le type.
    - si le type est PAD1, on met la longueur à 0 et le contenu à NULL et on retourne le TLV nouvellement alloué
    - sinon on lit 3 octets, et on calcule la taille
    - si le type est PADN, on décale le curseur du fichier en avant d'une valeur égale à la taille, le contenu est NULL et on retourne le TLV
    - sinon, on va allouer de l'espace pour le contenu, correspondant à la taille calculée.
    - si le type est COMPOUND, on renvoie le TLV, qui a pour l'instant un contenu vierge.
    - si le type est DATED, on lit 4 octets (la date) qu'on copie dans le contenu et on renvoi le TLV
    - pour tout autre type, on va lire des octets en boucle jusqu'à en avoir lu exactement le nombre décrit dans la taille, puis on le renvoie
  - on affiche le TLV récupéré avec `printTLV`, ou pour le cas HTML avec `tlvtoHTML`
  - en cas de DATED ou COMPOUND, on appelle la même fonction, mais avec des indices différents (la différence devrait valoir la valeur de la taille, à 4 près pour le DATED)
    - les TLV sont lus de la même manière et traités instantanément, mais on les ajoute au TLV parent (d'où le paramètre).
    - En fait, on n'affiche pas vraiment tout le COMPOUND/DATED : d'abord l'entête, puis les fils, uns par uns, pêle-mêle : on n'attend pas d'avoir eu tout.
  - Après avoir tout lu/affiché, on libère le TLV avec `freeTLV`, qui libère aussi le contenu des TLV différents de PAD1 et PADN.
  - on ferme le fichier

Ecriture

L'ajout de contenu se fait via le programme `writer`.

`writer <options> file [expressions]`

Par défaut, on va parcourir l'expression et, si elle est valide (voir *format*), on l'ajoute à la fin du fichier. Ensuite on passe aux éventuelles expressions suivantes (séparées par des espaces).
options :
* -d : pour ajouter automatiquement la date courante aux données ajoutées.
* -h : afficher l'aide

expression :
Le format est le suivant :
`<type>{<contenu>}`

où type est choisi parmi les lettres suivantes :
* t : TEXT {texte brut}
* p : PNG (nom du fichier)
* j : JPEG (nom du fichier)
* m : MP3 (nom du fichier)
* o : OGG (nom du fichier)
* v : OGV (nom du fichier)
* w : WEBM (nom du fichier)
* 1 : Pad1 (1 octet vide)
* n : PadN (n octets vides)
* d : dated (date-contenu) *où date est soit un entier UTC, soit calculée automatiquement (si autre chose de non vide est spécifié) *
* c : compound (c1-c2-c3)

Exemple :
Pour créer un Texte suivi d'un Compound composé d'un Texte, d'une image Jpeg Datée, et d'un Compound lui-même fait d'un Pad1 et d'un Texte :

`t{"BZH"}-c{t{"ma famille"}-d{32713-j{familleEnBretagne.jpeg}}-c{1-t{"tant de bonheur"}}}`

*notes : les guillemets ne sont pas obligatoires, mais permettent d'échapper les caractères d'espacement. Les espaces hors guillemets sont interdits. *
*Prendre garde aux caractères spéciaux du terminal : ',', '!', ... *

*Commentaires : il est sous-entendu dans le sujet que les ajouts se font à la fin : pas d'ajout possible en plein milieu. Nous avons compris ceci comme voulant dire :
"on ajoute du contenu, on ne modifie pas le reste du fichier".
L'ajout de contenu et la compression sont deux choses différentes : dans cette optique d'ajout sans modification, on ne supprime pas de pad1 ou padN si on ajoute un contenu en fin de fichier.
Nous avons voulu respecter la philosophie UNIX : Faire UNE chose, et la faire bien. Ainsi, un écrivain écrit : il ne supprimer/compacte/compresse pas. *

Implémentation

L'ajout de TLV à un fichier se fait grâce aux fichier "writer.c" et "writer.h".
- On s'assure d'avoir une commande correcte : options, fichier, expression(s)
- On appelle directement `addTLV(fichier, expression, date)`
  - si *date* est vrai, on modifie l'expression pour en faire un DATED de l'instant présent : "x" -> "d{t-x}"
  - extraction d'un TLV selon la chaîne donnée avec `exprtoTLV`.
    - analyse du 1er caractère : donne le type
    - si le type est PAD1, retour
    - si le type est PADN, analyse du second champ, qui donne la taille
    - si le type est TEXT, analyse du contenu brut (transcrit les caractères exacts si le format est respecté)
    - si le type est JPEG ou PNG, appel à `readimage` qui va ouvrir le fichier correspondant au nom écrit et copier les octets
    - si le type est DATED, séparation en deux avec `exprtodated` : d'un côté la date, de l'autre le sous-tlv, qui va être décrypté avec la même

fonction (appelée sur une substring)
    - si le type est COMPOUND, appel à `exprtocompound` qui va calculer le nombre de sous-tlv (séparés par '-') et pour chacun, appeler récursivement la fonction sur les substrings correspondantes tout en ajoutant les nouveaux TLV au compound
  - calcul de la taille (en prenant en compte celle des éventuels fils, et ajout de ces dits fils).
  - ouverture du fichier en écriture, et pose d'un verrou exclusif
  - création d'un fichier backup en cas de problème (soit on écrit toute l'expression, soit rien du tout)
    - `backup` et `restore` sont en fait de simple fork/exec sur cp.
  - écriture du TLV avec `writetlv`
    - si le fichier était vide, on ajoute les champs Magic et Version
    - sinon on vérifie l'entête, on doit rejeter tout fichier dont l'entête est incorrect
    - écriture du TLV
  - fermeture du fichier
- Eventuel rebouclage.

à tout moment, si une erreur non critique est detectée ou si le format est incorrect, les fonctions devant retourner un TLV retournent NULL, ce qui provoque l'arret du parsing et une tentative de restauration du backup.

Détails d'implémentation (fonctions annexes):

#Helpers

Les fichiers `helpers.c` et `helpers.h` définissent des éléments généraux, utilisables par tous les programmes du projet, mais sans être spécifiques aux dazibao : ils pourraient très bien être utilisés ailleurs.
`errsys` quitte sur un `perror`,
`min` et `max` calculent le minimum et le maximum entre deux entiers non signés.
`UTCtoString` converti un nombre de secondes UTC en une chaine de date au format "%F %A %H:%M" (2013/12/05 Thursday 14:51).
`printns` ne fait qu'imprimer n fois la chaine s (utilisée pour l'affichage textuel).
Nous bénéficions également des fonctions de conversion de boutisme suivantes : `bigendian`, `littleendian`, `localendian`, `otherendian`.

#Dazibao

Les fichiers `dazibao.c` et `dazibao.h` contiennent des élément généraux utiles pour chacun des programmes du projet, mais dépendemment de celui-ci.
Ils définissent notamment le type du tlv, `tlv_t` :

```
typedef struct tlv{
  tlv_type type;
  unsigned int length;
  byte * content;
} tlv_t;
```

où`tlv_type` est un élément d'une enum {PAD1, PADN, TEXT, PNG, JPEG, COMPOUND, DATED}.

Y sont également spécifiées les fonctions de conversion entre un tableau de 3 octets et une taille (`computelength` et `arrayedlength`),
une fonction pour vérifier la version (numéro magic = 53 et version = 0)

`checkversion`, une fonction d'ouverture de fichier avec un verrou spécifique (`lockopen`).
 Il y a également deux fonctions de "modification" de TLV composés (Dated et Compound) : ajout de sous-tlv (`appendTLV`) ou extraction de sous-tlv (`extractTLV`).
Enfin, comme nous utilisons l'allocation dynamique, il est nécessaire de libérer la mémoire : `freeTLV` libère le contenu du TLV, puis le TLV.

Suppression

La suppression se fait avec le programme `deleter`.

`deleter <nom de fichier>`

Le programme parcourt le fichier en mode interactif du début jusqu'à la fin (sauf si l'utilisateur veut arrêter avant). Nous pouvons supprimer n'importe quelle TLV, y compris celles qui se trouvent à l'intérieur des types composés (compound ou dated). Pour afficher la TLV courante, nous utilisons la partie textuelle de Reader. La TLV courante est entourée par des '*' pour faire une distinction dans le cas des types composés.

```
********************************************************************************
_
type = dated | longueur = 36998
date: 2013-11-06 Wednesday  13:28 (154565202)
read a TLV type 4
_
----type = image JPEG | longueur = 36990
----(image JPEG)
********************************************************************************
Possible actions:
del-delete
n-next TLV
jump <number>-jump of <number> TLVs
exit-exit :)
e-explore the 'dated'

Choose action:
e
```

```
_
type = dated | longueur = 36998
date: 2013-11-06 Wednesday  13:28 (154565202)
********************************************************************************
_
----type = image JPEG | longueur = 36990
----(image JPEG)
********************************************************************************
Possible actions:
del-delete
n-next TLV
jump <number>-jump of <number> TLVs
exit-exit :)

Choose action:
```

Les actions possibles sont affichées en fonction de contexte. L'utilisateur peut : supprimer une TLV, explorer un type composé (quand on veut supprimer une TLV à l'intérieur d'une autre TLV), passer à une TLV suivante, faire un « saut » d'un certain nombre de TLV (pour permettre une navigation rapide dans le fichier), quitter le programme. Les « sauts » se font à une profondeur donnée, c'est-à-dire si par exemple on se trouve à l'intérieur d'un compound, « jump 2 » fera un saut de 2 TLV dans ce compound. De même pour le passage à une TLV suivante.

Implémentation :
Au début, le programme prend un verrou exclusif sur le fichier.
Nous utilisons une boucle semblable à celle dans Reader. Si l'utilisateur a choisi de supprimer la TLV courante, on décale le offset à l'endroit où se trouve le champ « type » (l'entier « left » dans la fonction) de la TLV courante, on le remplace par PADN et on remet l'offset à l'endroit précédant.
Si l'utilisateur a choisi de faire un saut d'un certain nombre de TLVs, nous avons un compteur qui indique le nombre de TLV à ignorer (on les affiche quand même mais on ne propose pas d'actions à l'utilisateur).
Si l'utilisateur veut explorer un type composé non vide, on procède à un appel récursif sur la portion correspondant au contenu du champ à explorer.

Compaction

Pour compacter un fichier, il suffit de lancer le programme compacter.

`compacter <nom de fichier>`

La compaction supprime tous les PAD dans tout le fichier,y compris à l'intérieur des types composés(compound et dated). Le programme affiche la taille actuelle du fichier, les 'offset' des PAD et la nouvelle taille du fichier, si le fichier a été modifié. Sinon le programme informe l'utilisateur que le fichier est déjà compacté.

Fonctionnement/implémentation
Le programme prend un verrou exclusif sur le fichier.
Comme suggéré dans l'énoncé, nous faisons un déplacement du contenu sur place et on finit par un ftruncate (s'il y avait au moins un PAD).
Nous avons une variable globale offset_write qui indique la position sur laquelle on devrait écrire. Cette variable est modifiée lors de l'exécution du programme. La valeur de cet offset_write est forcément inférieure ou égale à « l'offset de lecture ».
Au début, on procède à la lecture du fichier. Si on trouve un PAD, on met le offset_write à la position de ce PAD (c'est à partir de cette position qu'on devrait commencer le décalage du contenu). Les prochaines TLV autres que les PAD seront écrites dans le fichier à l'offset indiqué par offset_write (qui sera modifié au fur et à mesure).
Dans le cas des types composés, nous devons modifier leur longueur, si leur contenu a été compacté. L'algorithme marche de manière récursive. La fonction loop_compacter (boucle semblable à celles utilisées dans d'autres parties du projet) renvoie le nombre d'octets autres que ceux des PAD. Ceci permet d'indiquer à la « TLV parent » la taille du nouveau contenu. Nous connaissons l'offset des champs « length » des TLV en question grâce au paramètre p_offset_length de la fonction readtlv_compacter (qui fait en même temps l'écriture,donc qui connaît la position de ce champ après décalage). Il suffit maintenant do comparer l'ancienne longueur avec la longueur renvoyée par l'appel récursif. Si cette valeur est différente, on fait un lseek à offset_length, on écrit la nouvelle longueur et on rétablit l'ancien offset (avec un lseek aussi).

Notifications

Le serveur de notifications est le programme `dzbNotify`.

`dzbNotify <nom de fichier contenant des chemins de fichiers dazibao>`

dzbNotify est le programme serveur. Il n'a pas besoin de interaction avec l'utilisateur. Il surveille les événements sur les fichiers Dazibao et signale ces événements aux clients.

Implémentation :

dzbNotify est un programme multi-thread. Un thread gère l'addition et la suppression de clients, un autre thread surveille des changements dans les fichiers et envoie des messages aux clients. Les fichiers sont surveillés grâce au système de notifications de Linux. Le programme utilise un pthread_mutex pour synchroniser l'accès aux variables partagées. Nous avons choisi d'utiliser des listes chaînées : une pour stocker les chemins vers les Dazibao et une autre pour stocker les descripteurs qui serviront à communiquer avec les clients. La suppression des clients se fait grâce au system call *poll() : q*uand un client a terminé, *POLLRDHUP event* est envoyé et le descripteur du client est supprimé de la liste.
Un client est informé d'un changement lorsque le contenu d'un fichier a été modifié et le programme modifiant le fichier a fini de travailler sur celui-ci. Pour ceci nous avons eu l'idée suivante : quand l'événement IN_MODIFY (contenu modifié) arrive pour un fichier, nous avons une variable (qui nous sert d'un booléen) qui mémorise ceci pour ce fichier. Quand l'événement IN_CLOSE_WRITE (c'est-à-dire qu'un programme qui a ouvert le fichier en écriture a fini de travailler sur le fichier) a lieu, on informe les clients du changement si et seulement si avant nous avons reçu un IN_MODIFY pour ce fichier. Nous avons opté pour cette solution pour ne pas recevoir des notifications 'redondantes' (par exemple si quelqu'un écrit un mp3 dans un fichier, on n'a pas besoin d'avoir 5 notifications).