EPFL CS212 : ImgStore – Système de fichiers orienté images — 13 : garbage collecting

E. Bugnion & J.-C. Chappelier EPFL

Rev. 2021.04.22 / 1

Table des matières

Introduction

Dans ce dernier sujet (la dernière semaine vous étant laissée pour finaliser ou ajouter quelques extensions optionnelles à votre projet), vous allez rajouter une dernière commande à votre utilitaire de ligne de commandes `imgStoreMgr` : la commande `gc` (« *garbage collecting* »).

Cette commande fait un certain « nettoyage » de la base d'images (et non pas de la mémoire gérée, p.ex. comme en Java). Elle va collecter et supprimer divers « trous » dans un fichier au format `imgStore` sur le disque. En effet, puisque la commande `delete` n'efface pas directement les images et ne réduit pas la taille du fichier, mais rend simplement leur contenu inaccessible, certaines parties du fichier sont en fait inutiles. `gc` va les supprimer, sans, bien sûr, rien modifier au contenu effectivement utilisé (le résultat des commandes `list` et `read`, par exemple, doivent rester inchangés).

La commande `gc` aura donc comme conséquences de :

— ne pas changer du tout le fichier s'il y n'y a pas de « trous » (et, en toute rigueur, ni de doublon de « petites » images ; mais c'est un détail ici) ; elle est en particulier idempotente ;
— ne pas changer le contenu visible/utilisé de la base d'images ;

— ne pas changer la taille du tableau de `metadata` ;
— réduire au maximum la taille du fichier s'il existe des images non utilisées ;
— peut-être supprimer certains des doublons de « petites » images (« *small* » ou « *thumnail* ») éventuellement existantes (mais pour simplifier au niveau de ce projet, ceci ne sera qu'un effet secondaire de la méthode simple proposée et non pas un nouvel algorithme, ni un but en soi).

L'objectif principal de cette semaine est donc d'implémenter `gc` tout en :

— réutilisant des fonctions existantes autant que possible ;
— écrivant aussi peu de lignes de code que possible ;
— modularisant votre code autant que possible ;
— et profitant bien de cet exercice en vous amusant, puisque c'est le dernier `:-)`.

Un objectif secondaire est, si ce n'est pas déjà fait, de mettre à jour votre utilitaire « *command line manager* » `imgStoreMgr` par rapport aux modifications des dernières semaines.

Matériel fourni

Comme les semaines passées, le travail de cette semaine se construit sur tout votre travail des semaines précédentes, mais nous vous fournissons un fichier de test (le même que le « *feedback* »).

Travail à faire

Modifications préliminaires

Si ce n'est pas encore fait, reporter la modification faite en semaine 9 à la commande `do_list()` (ajout d'un mode) à votre « *command line manager* ».

Garbage collection

La commande `gc` est similaire à tous les autres commandes dans le « *command line manager* ». Elle prend deux arguments de ligne de commande : 1. le nom du fichier `imgst` à traiter ; 2. un nom de fichier temporaire (qui *a priori* n'existe pas encore ; mais s'il existe sera simplement écrasé).

Pourquoi ce fichier temporaire ?
Pour plusieurs raisons (simplification du sujet, mais aussi garantie d'intégrité en cas d'échec), la commande `gc` ne fait pas le « *garbage collecting* » sur place dans le fichier d'origine, mais en *copiant* le contenu. Cette copie nécessite un fichier transitoire temporaire dont il est préférable de donner le nom ici dès le départ : l'ingénieur système décidant de lancer un « *garbage collecting* » sur une (grosse) base d'images choisit ainsi où il souhaite que son fichier temporaire soit stocké (et accessoirement comment il va s'appeler, pour éventuellement le retrouver, en cas de crash par exemple).

Un exemple de commande `gc` serait donc :

```
./imgStoreMgr gc my_favorite_file.imgst /tmp/gc_20210523082000.imgst
```

Le cœur de cette commande devra être implémenté dans une fonction `do_gbcollect()` à mettre dans un nouveau fichier `imgst_gbcollect.c`. La fonction `do_gbcollect()` retourne un `int` correspondant à un code d'erreur, `ERR_NONE` en cas de succès et prend 2 arguments :

1. le nom du fichier `imgst` original, à nettoyer ;
2. le nom du fichier, qui sera utilisé temporairement pour la nouvelle version, « nettoyée ».

Durant l'essentiel de la phase de « nettoyage », il y aura donc deux fichiers avec les mêmes images : le fichier original (non modifié) et le fichier temporaire, en cours de création/modification, version « nettoyée » du fichier d'origine.

Une fois la nouvelle version totalement terminée, le fichier temporaire est alors renommé avec le nom d'origine. Pour cette rocade, nous vous demandons d'utiliser simplement les fonctions `remove()` et `rename()` de `stdio.h` ; une version complète et robuste étant beaucoup trop compliquée à ce niveau (pour ceux que cela intéresse, voir par exemple le code source de `mv` et de `copy`).

Note technique : À noter donc, qu'en raison de ces simplifications (fonctions `remove()` et `rename()`), le fichier temporaire doit nécessairement être sur le même système de fichiers (*filesystem*) que le fichier de départ. Il se peut donc que, sur certaines machines, la commande ci-dessus avec un fichier temporaire sur `/tmp` ne fonctionne pas correctement, typiquement lorsque `/tmp` est un autre *filesystem* que celui de votre fichier de départ. Dans ce cas, testez simplement votre programme avec des commandes du genre :

```
./imgStoreMgr gc my_favorite_file.imgst tmp.imgst
```

(où les deux fichiers sont donc sur le même *filesystem*).
[fin de note]

Pour implémenter la fonction `do_gbcollect()`, réutilisez le plus possible les fonctions existantes. Dans le cas idéal, cela inclut `do_create()`, `do_read()`, `do_insert()` et `lazily resize()`. Vous pouvez faire toutes les modifications dont vous avez besoin (et uniquement celles qui sont nécessaires) avec et sans changer la sémantique des ces autres commandes.

Attention en particuliers aux choix que vous avez fait pour `do_create()` : quel mode d'ouverture du fichier ? le fichier est-il fermé ou non par `do_create()` ?

La fonction `do_gbcollect()` doit conserver toutes les images utiles, y compris les « petits formats » (« *small* » ou « *thumnail* ») déjà présents (sauf doublons, bien sûr ; et sans en rajouter).

En cas d'erreur sur `remove()` ou `rename()`, la fonction retournera simplement `ERR_IO`.

Enfin, la fonction `do_gbcollect()` doit fermer tous les flots qu'elle a ouvert.

Intégration dans l'utilitaire

Comme d'habitude, il faut ajouter la fonction correspondant à `do_gbcollect()` dans le « *command line manager* » :

1. rajouter une fonction `do_gc_cmd()` qui, si la commande `gc` a été donnée sur la ligne de commande, va au moins appeler la fonction `do_gbcollect()` ;
2. complétez l'aide pour ajouter l'information suivante :

```
gc <imgstore_filename> <tmp imgstore_filename>: performs garbage collecting on imgSto
```

Faites enfin, bien sûr, les modifications nécessaires sur votre `Makefile`.

Tests

Pour vérifier votre implémentation, vous pouvez :

— insérer plusieurs images (y compris des doublons. . .) ;
— faire quelques `read` pour ajouter des « petites » images ;
— faire `list` pour vérifier ; utiliser la commande `ls` pour déterminer la taille du fichier ;
— effacer quelques images ;
— encore faire `list` et vérifier la taille du fichier ; normalement, elle n'a pas changé ;
— faire `gc` ;
— vérifier que la sortie de `list` n'a pas changé, mais que la taille du fichier est réduite.

Vous pouvez également utiliser/vous inspirer du fichier `13.test-gc.sh` fourni (qui est celui utilisé dans le « *feedback* »).

Rendu

Le code à ce stade (c.-à-d. tout le travail depuis la semaine 4, éventuellement corrigé des retours de correction) constitue le **rendu final** de la partie projet de ce cours. Il est à rendre avant le **dimanche 6 juin 23:59**. Aucune extension de délai d'aucune sorte ne saurait être accordée.

Pour le rendre, il n'y aura rien à faire de plus que d'avoir bien ajouté (`git add`), validé (`git commit`) et transmis (`git push`) toutes vos dernières versions de tous *vos* fichiers sources `.c` et `.h`, ainsi que le `Makefile`. Ce sera en effet la version se trouvant dans votre branche principale (`master`) le lundi 7 juin à 00:00 qui sera considérée comme votre rendu final. Il n'y a donc **pas** de « submit » à faire cette fois ci.

Enfin n'oubliez pas de mettre une dernière fois à jour votre fichier `time.csv` et d'ajouter un fichier `README.md` comme ce sera expliqué dans le sujet de la semaine prochaine (déjà en ligne).

Toute bonne finalisation !