EPFL CS212 : ImgStore – Système de fichiers orienté images — 08 : arguments de la ligne de commande et déduplication des images

E. Bugnion & J.-C. Chappelier EPFL

Rev. 2021.01.07 / 1

Table des matières

Projet de Programmation Système – CS212 – 2021

ImgStore – Système de fichier orienté images

08 : arguments de la ligne de commande et déduplication des images

Introduction

Cette semaine consiste en deux objectifs distincts (pensez à vous répartir le travail) :

— traiter de façon plus portable les arguments de ligne de commande en utilisant la fonctionnalité des pointeurs sur fonctions ; nous en profiterons pour ajouter des options à la commande `create` ;
— incorporer les éléments qui permettront la dé-duplication des images sauvegardées (ne pas dupliquer des images identiques).

Note/Rappel : si vous travaillez en avance sur le rendu n'oubliez pas de faire un `make submit1` avant de « polluer » votre premier rendu avec du travail en avance. En cas de doute/difficulté, demandez à un assistant (ou un enseignant).

Matériel fourni

Nous ne fournissons cette semaine que 2 programmes de test car, comme précédemment, vous construisez votre projet sur la base des semaines précédentes.

A noter que `util.c` déjà fourni par le passé, offre deux fonctions outils (`atouint16()` et `atouint32()`) permettant de convertir une chaîne de caractères contenant un nombre en sa valeur en `uint16` ou `uint32`. Nous vous encourageons à utiliser ces deux fonctions pour convertir les chaînes de caractères des arguments de ligne de commande. Elle gèrent les différents cas d'erreurs en cas de conversion d'un nombre non valide, ou d'un nombre trop grand pour le type spécifié (p.ex. en essayant de convertir 1000000 en un nombre de 16 bits). Elles retournent 0 dans ces cas. Utilisez les pour implémenter correctement votre code.

Travail à faire

Analyse (« *parsing* ») des arguments de ligne de commande

Unification de la signature des fonctions de commande Regardez la fonction `main()` dans la version actuelle de `imgStoreMgr.c` : elle contient à la fois le *parsing* des commandes (par exemple, « `list` », « `create` », « `delete` », etc.) ainsi que le *parsing* des arguments de ces fonctions (par exemple, `do_delete_cmd()` prend 2 arguments, mais `do_create_cmd()` n'en prend qu'un seul). Cette double logique rend le code difficile à maintenir et à faire évoluer (ajout de nouvelles commandes). Par exemple, si l'on décide de changer les arguments d'une commande particulière, il faut à la fois changer l'implémentation de de cette commande, ainsi que changer l'appel à cette fonction dans le `main()`.

L'objectif de cette semaine est d'unifier la gestion des commandes reçues en utilisant les pointeurs sur fonction. On pourrait même imaginer (extension hors du projet, non demandée) transformer le programme en un interpréteur de commandes (tournant sans fin jusqu'à la commande `quit`).

Tout d'abord, vous devez changer la signature de toutes les fonctions `do_COMMAND_cmd()` (et `help()`).
Actuellement, elles ont chacune une signature propre. Pour simplifier la logique, nous devons uniformiser leur signature. La solution est toute simple : plutôt

que de *parser* les arguments dans `main()`, il suffit de déléguer le *parsing* à ces fonctions et de passer tous les arguments de ligne de commande à ces fonctions :

```
int do_COMMAND_cmd(int args, char* argv[])
```

Donnez aussi ce même prototype à la fonction `help()`.

Maintenant, `main()` peut simplement appeler de manière identique chaque fonction qui gère une commande.

Utilisation de tables pour simplifier la logique La version originale de `main()` utilise une approche « *if-then-else* » pour traiter des différentes commandes possibles. Cette approche rend plus difficile le rajout de nouvelles commandes (qui arriveront dans les semaines suivantes !) puisqu'il faut à chaque fois rajouter un nouveau cas. Elle rend aussi le code rapidement illisible en cas de nombreuses commandes.

La seconde modification proposée ici a pour but d'éliminer ces tests répétitifs en les remplaçant par une simple boucle. L'objectif de cette boucle est tout simplement de rechercher la commande parmi une liste de commandes possibles (pour le moment « *list* », « *create* », « *help* », et « *delete* ») et d'appeler la fonction correspondant à chaque commande.

Nous allons donc mettre les différentes fonctions `do_COMMAND_cmd()` précédemment unifiées (et `help()`) dans un tableau. On en profitera pour associer les noms des commandes avec leur fonctions respectives (p.ex. la chaîne `"list"` avec la fonction `do_list_cmd()`).

Pour cela, dans `imgStoreMgr.c` : 1. définissez un type `command`, pointeur sur fonction telles que celles unifiées ci-dessus ; 2. définissez un type `struct command_mapping` contenant une chaîne de caractères (constante) et une `command`.

Utiliser ensuite ces définitions pour créer un tableau nommé `commands` associant les commandes « *list* », « *create* », « *help* », et « *delete* » aux fonctions correspondantes.

Finalement, vous réécrivez le `main()` en utilisant ce tableau à l'intérieur d'une boucle. Lorsque la bonne commande est trouvée, il vous suffit d'appeler la fonction pointée dans l'entrée correspondante du tableau, en passant l'ensemble des arguments de ligne de commande.

Par exemple, si vous appelez le programme

```
./imgStoreMgr list imgst_file
```

alors, votre code doit appeler `do_list_cmd()` avec les paramètres suivants: `argc` (qui, après le `argc--` fourni, vaut 2 dans cet exemple) et `argv` (qui, après le `argv++` fourni, vaut `{ "list", "imgst_file" }` dans cet exemple).

Votre code doit correctement traiter le cas où la commande n'est pas définie : appeler `help()` et retourner `ERR_INVALID_COMMAND` dans ce cas-là.

Votre code peut tout à fait faire l'hypothèse que toutes les commandes du tableau `commands` sont distinctes. Réfléchissez-y et simplifiez en conséquence.

Gestion des paramètres

Maintenant que l'infrastructure de *parsing* est plus flexible, il est temps de déplacer le traitement des arguments dans les fonctions `do_COMMAND_cmd()` correspondantes.

Ceci fait, nous allons ensuite reprendre la commande `create` afin d'y introduire une plus grande flexibilité. Dans cette étape, vous devez rajouter les arguments suivants à la fonction `create`:

- `-max_files <MAX_FILES>`: le nombre d'images maximal dans un seul `imgStore` ;
- `-thumb_res <X_RES> <Y_RES>`: résolution des images « thumbnail » ;
- `-small_res <X_RES> <Y_RES>`: résolution des images en format « petit ».

Dans le premier cas, l'argument est un nombre entier positif de 32 bits. Dans les autres cas, les résolutions sont des nombres entiers positifs de 16 bits.

Ces trois nouveaux arguments de la commande `create` sont optionnels. Si l'argument n'est pas spécifié dans la ligne de commande, vous devez continuer à utiliser comme valeur par défaut les valeurs actuellement utilisées par `do_create_cmd()` (par exemple `max_files = 10`).

Ces trois nouveaux arguments auront également des valeurs maximales acceptables possibles telles qu'indiquées ci-dessous (dans `help()`).

Changer `help` Changez la commande `help` afin de refléter les nouveaux paramètres de cette façon :

```
> ./imgStoreMgr help
imgStoreMgr [COMMAND] [ARGUMENTS]
  help: displays this help.
  list <imgstore_filename>: list imgStore content.
  create <imgstore_filename> [options]: create a new imgStore.
      options are:
          -max_files <MAX_FILES>: maximum number of files.
                                  default value is 10
                                  maximum value is 100000
          -thumb_res <X_RES> <Y_RES>: resolution for thumbnail images.
                                  default value is 64x64
                                  maximum value is 128x128
          -small_res <X_RES> <Y_RES>: resolution for small images.
                                  default value is 256x256
```

```
                                    maximum value is 512x512
  delete <imgstore_filename> <imgID>: delete image imgID from imgStore.
```

Pour vos tests, pensez à « rapatrier » `provided/tests/helptext_week08.sh` comme le nouveau `helptext.sh` dans votre `done/tests`.

Parsing des arguments de la commande `create` Si ce n'est pas encore fait, la dernière modification à apporter à `do_create_cmd()` est de *parser* correctement tous ses arguments – les arguments obligatoires ainsi que les arguments optionnels.

Votre solution doit avoir la structure suivante :

- commencez par récupérer l'argument obligatoire (`<imgstore_filename>`)
- itérer ensuite sur `argv` ;
- à chaque itération, déterminer tout d'abord si il s'agit d'un argument optionnel acceptable (`-max_files`, `-thumb_res` ou `small_res`) ;
- si c'est le cas, vérifier s'il reste encore assez de paramètres pour les valeurs correspondantes (au moins un pour `-max_files` et au moins 2 pour les deux autres) ; si ce n'est pas le cas, retourner `ERR_NOT_ENOUGH_ARGUMENTS` ;
- convertissez ensuite le ou les deux paramètres suivants dans le bon type ; vérifier que la valeur est correcte (non nulle ni trop grande) ; en cas d'erreur, retournez soit `ERR_MAX_FILES` (pour `-max_files`), soit `ERR_RESOLUTIONS` ;
- si il ne s'agit pas d'un argument optionnel, retourner l'erreur `ERR_INVALID_ARGUMENT`.

A noter :

- les arguments optionnels pourraient être répétés, par exemple `-max_files 1000 -max_files 1291` ; dans ce cas, seule la dernière valeur fait foi ;
- l'argument obligatoire ne peut pas être répété.

Après avoir traité de tous les arguments et paramètres, le reste du code ne change pas, si ce n'est qu'évidemment les fichiers de type `imgStore` doivent être créés avec les valeurs reçues en paramètres !

Dé-duplication des images

Le second composant de la semaine concerne la dé-duplication d'images, pour éviter qu'une même image (même contenu) soit présente plusieurs fois dans la base. Au niveau d'un réseau social, ce type d'optimisation permet de gagner beaucoup de place (et de temps).
Pour ce faire, vous devez écrire une fonction `do_name_and_content_dedup()` à définir dans un nouveau fichier `dedup.c` (et à prototyper dans `dedup.h`).

Cette fonction retourne un code d'erreur (`int`) et prend deux arguments (dans cet ordre) :

— un fichier `imgStore` précédemment ouvert ;
— un index (type `uint32_t`ici) qui spécifie la position d'une image donnée dans le tableau `metadata`.

Dans le fichier `dedup.c`, implémentez ensuite cette fonction comme suit.

Pour toutes les images valables dans le fichier `imgst_file` (autre que celle à la position `index` et par positions croissantes) :

— si le nom (`img_id`) de l'image est identique à celui de l'image à la position `index`, retourner `ERR_DUPLICATE_ID` ; il s'agit ici de garantir que la base d'image ne contient pas deux images avec le même indentifiant interne ;
— (puis,) si la valeur SHA de l'image est identique à celle de l'image à la position `index`, on peut alors éviter la duplication de l'image à la position `index` (pour toutes ses résolutions).

Pour dé-dupliquer, il faut modifier les métadonnées à la position `index`, pour y référencer les attributs de la copie trouvée (ses trois offsets et ses deux/trois tailles ; notez que la taille d'origine est forcément la même).

Note : on ne modifie surtout pas le nom (`img_id`) de l'image à la position `index` : ce ne sont que les contenus qui sont dé-dupliqués ; on aura deux images de nom différent, mais pointant vers le même contenu.

Si l'image à la position `index` n'a pas de doublon de contenu, mettez son offset de `RES_ORIG` à 0.
Si l'image à la position `index` n'a pas de doublon de nom (`img_id`), retourner `ERR_NONE`.

Suggestion : comme première étape, écrivez une fonction qui compare deux valeurs de SHA pour déterminer si la valeur est identique.

Tests

Pour le traitement de ligne de commande, testez différentes combinaisons (valables et non-valables). Une esquisse de ces tests est fournie dans `provided/tests/08.test-create.sh`.

La dé-duplication ne peut pas être facilement testée de bout en bout cette semaine ; il faudra pour cela attendre la semaine prochaine (`do_insert()`). Pour l'instant, assurez-vous déjà que votre code compile correctement. On peut ensuite tester artificiellement la fonctionnalité de `do_name_and_content_dedup()` sur des données artificielles construites spécifiquement pour cela. Regardez à ce sujet les tests unitaires fournis dans `provided/tests/unit-test-dedup.c`.

Rendu

Vous **n'**avez **pas** à rendre de suite le travail de cette semaine dans le premier rendu à faire d'ici dimanche soir. Par contre, le travail de cette semaine sera à

rendre à la fin de la semaine 11 (délai : le dimanche 16 mai 23:59) en même tant que le travail de la semaine 9 (et la révision de votre code des semaines 4 à 7).

Ceci dit, nous vous conseillons comme toujours de marquer par (au moins) un commit le travail correspondant à cette semaine (vous pouvez en faire d'autres avant, bien sûr !), et de (continuer à) travailler régulièrement.

Et n'oubliez pas de faire le rendu (`make submit1`) de la semaine passée (semaines 4 à 7) avant ce dimanche soir.