SmartFolder

Maxime Lovino Thomas Ibanez

27 janvier 2017

1 Introduction

Nous avons réalisé notre programme SmartFolder en 3 étages principaux. Un bloc de structure de données et utilitaires qui servent à toutes les autres parties du programme. Un bloc moteur, qui regroupe le Parser ainsi que le moteur de recherche et le créateur de liens. Et tout en haut, un bloc qui éxecute, il s'agit donc du main du programme, ainsi que de la partie Daemon qui va tourner en tâche de fond et lancer la recherche incrémentale.

Vous pouvez trouver en dernière page de ce document, un schéma de cette architecture.

1.1 Data Structures

1.1.1 List

Nous avons une structure de liste qui sert à stocker la liste des résultats d'une recherhe (liste de fichiers). Nous définissions des fonctions permettant d'agréer plusieurs listes au travers d'une union, intersection, etc... correspondant aux opérations booléennes classiques.

1.1.2 HashSet

Nous avons une structure de table de hachage permettant de stocker les matches actuels présent dans le dossier, cela nous permet lors du prochain run de la recherche de tester de façon rapide (recherche en $O(1)$ en général et maximum en $\Theta(n)$).

1.1.3 Stack

Nous utilisons une structure de Pile pour stocker les résultats d'une recherche particulière (on stocke un pointeur de la liste des résultats) pour ensuite les rassembler lorsque nous trouvons un opérateur booléen. (principe de l'évaluation d'une expression polonaise inverse)

1.2 Utils

1.2.1 Logger

Nous avons réalisé un Logger pour pouvoir afficher sur la console des messages de notre programme, pour ceci nous définissons 3 niveaux de severité pour un message :

- 0 : DEBUG, ce mode est utilisé en debug uniquement, beaucoup trop verbose pour la production
- 1 : INFO, un message d'information pour l'utilisateur, comme un fichier trouvé par exemple.
- 2 : WARNING, un message de warning, par exemple une erreur dans notre programme durant son execution mais qui n'empêche pas le programme de continuer
- 3 : FATAL, un message d'erreur, lorsqu'une erreur se produit qui fait crasher le programme entièrement

Dans le fichier `Logger.h` on peut changer la valeur du `#define LOG_LEVEL X` par le niveau minimal que l'on veut. C'est-à-dire que si nous définissons 1 ici, le logger affichera uniquement les messages de niveau 1 et 2. En version release, le niveau est configuré à 1.

Dans notre code nous pouvons utiliser la fonction `void logMessage(int level, const char* format, ...);` pour logger un message en spécifiant son niveau et en utilisant une syntaxe similaire à `printf(...)` ensuite.

1.2.2 SysCall

Nous avons un fichier regroupant des fonctions wrappers pour les appels systèmes que nous utilisons dans les différentes parties de notre programme. De ce fait, si nous voulons porter notre programme sur un autre système, nous avons juste à modifier le contenu de ces fonctions pour adapter ces appels systèmes.

Les valeurs de retour des appels systèmes ont été standardisées pour retourner un booléen, c'est-à-dire un 1 en cas de réussite et un 0 dans le cas contraire.

1.2.3 SysFile

Dans cette partie, nous allons créer les fichiers qui nous serviront à retrouver une instance de SmartFolder déjà lancée sur la machine. Nous avons également des fonctions pour supprimer ces fichiers lors de la suppression du SmartFolder. Ces fichiers sont stockés dans `/tmp/`.

Il s'agit de deux fichiers, l'un ayant pour nom le nom du SmartFolder et contenant son PID, l'autre ayant pour nom le PID et contenant le chemin complet du SmartFolder.

1.3 Engine

1.3.1 Search

Dans la partie recherche nous avons la fonction qui va effectuer une recherche d'un certain type avec un certain argument de recherche dans un dossier et va retourner une liste des résultats de cette recherche. Nous définissons les types de recherche possible dans une énumération. Nous passons à la fonction le dossier à chercher, le type de recherche et l'argument de la recherche en question.

1.3.2 Parser

Ici nous avons la grande partie du programme, avec principalement la fonction

```
int evaluateAndSearch(char** expression, int exprLen, char* folder, HashSet** result)
```

qui va analyser la query passée en paramètre au programme (sous forme polonaise inverse) et l'éxecuter en stockant les résulats dans le HashSet `result`.
Nous avons également d'autres fonctions permettant de vérifier si une fragment de recherche est valide, détecter des opérateurs booléens, recupérer l'UID d'un nom d'utilisateur ainsi que vérifier que le chemin d'accès du dossier est valide. Pour cette dernière, nous utilisons une RegExp en utilisant `Regex.h` pour computer la regex.

1.3.3 Linker

Le Linker sert à créer un lien symbolique vers un fichier dans le dossier de destination, il va s'occuper de créer un symlink avec le nom du fichier et gérer les collisions de noms le cas échéant (par exemple `./a.txt` et `./abc/a.txt` vont donner des symLink `a.txt` et `a.txt(1)`)

1.4 Daemonizer

Le Daemonizer va s'occuper de faire tourner en arrière-plan notre programme et de merger les nouveaux résultats de recherche avec les résultats déjà liés précedemment. Cela comprendra également la suppression de fichiers qui n'existent plus dans le dossier d'origine, ou qui ne matchent plus la recherche.

1.5 SmartFolder

Partie principale du programme, c'est ici qu'on s'occupera de prendre les arguments de l'utilisateur pour la création d'une instance de SmartFolder, ainsi que pour la suppression d'un SmartFolder existant.

2 Utilisation du programme

Nous pourrons appeler notre programme de deux manières, premièrement pour créer un SmartFolder :

```
SmartFolder <linkDirectory> <searchDir> [searchQuery]
```

ou pour supprimer un SmartFolder existant :

```
SmartFolder -d <linkDirectory>
```

2.1 Syntaxe de recherche

Au niveau de la `[searchQuery]` les arguments suivant pourront être utilisés

- `--name <name>`
- `--size [-+]<size>`[KM/G/T]/
- `--dateStatus [-+]<YYYY-MM-DD>`
- `--dateModified [-+]<YYYY-MM-DD>`
- `--dateUsed [-+]<YYYY-MM-DD>`
- `--uid <loginName>`
- `--gid <groupName>`
- `--perms <octal>`

ainsi que les opérateurs booléens classiques sous cette forme

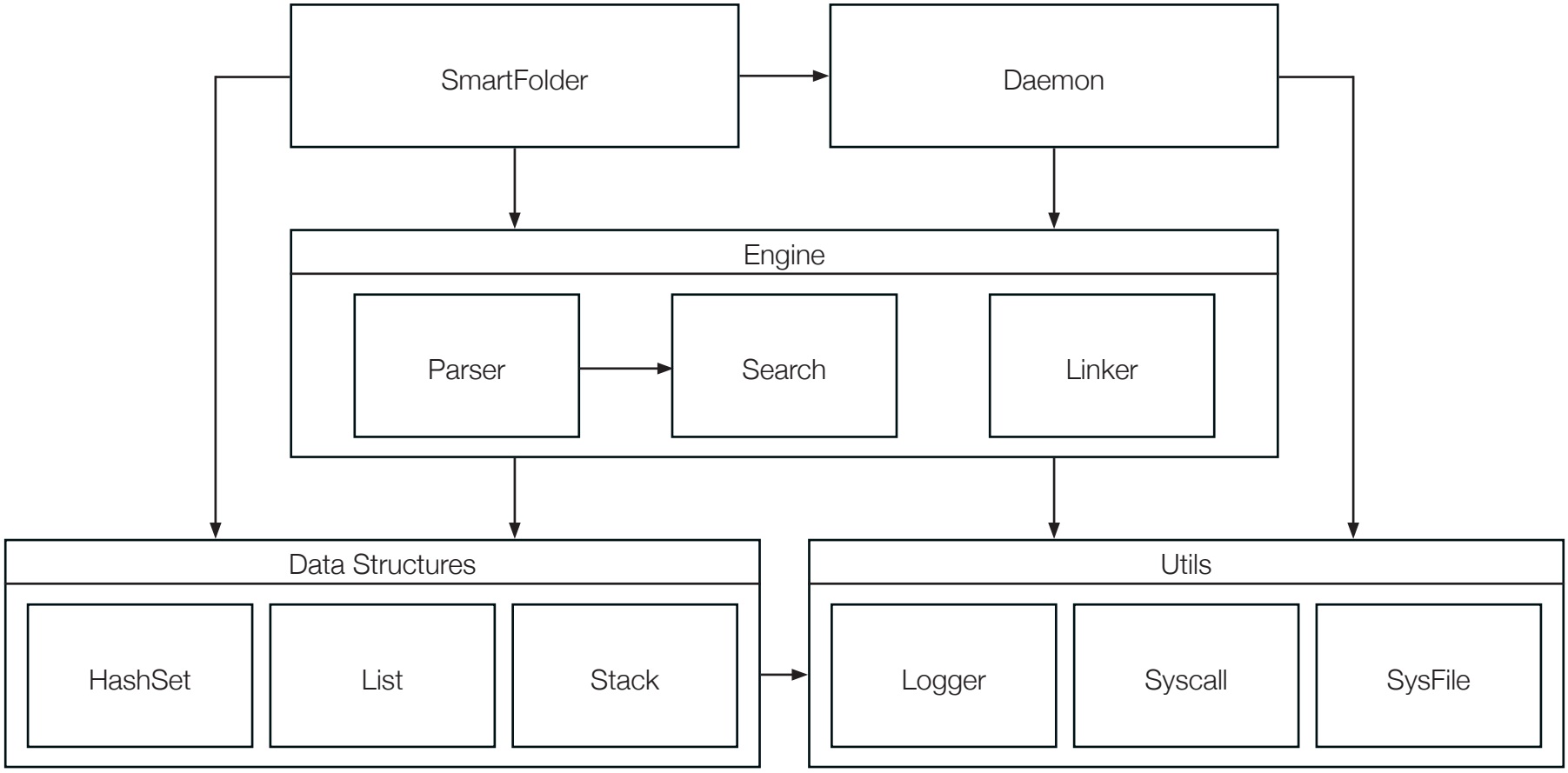
- `AND/and`
- `OR/or`
- `XOR/xor`
- `NOT/not`

La query sera sous la forme polonaise inverse, ce qui donne par exemple pour chercher tous les fichiers contenant `toto` dans le nom ET ayant comme permissions 777 OU les fichiers du user `lovino`.

```
--name toto --perms 777 AND --uid lovino OR
```

3 Conclusion

En conclusion, nous avons énormément apprécié travailler sur ce projet, qui est à la fois utile et intéressant. Nous avons pu découvrir beaucoup d'aspects des appels systèmes et la puissance du monde UNIX. Ce projet nous a servi également à encore amélioré nos connaissances du langage C, ainsi qu'à mieux gérer un projet d'une certaine envergure sur plusieurs mois.


SmartFolder
Daemon
Engine
Parser
Search
Linker
Data Structures
HashSet
List
Stack
Utils
Logger
Syscall
SysFile