# SmartFolder

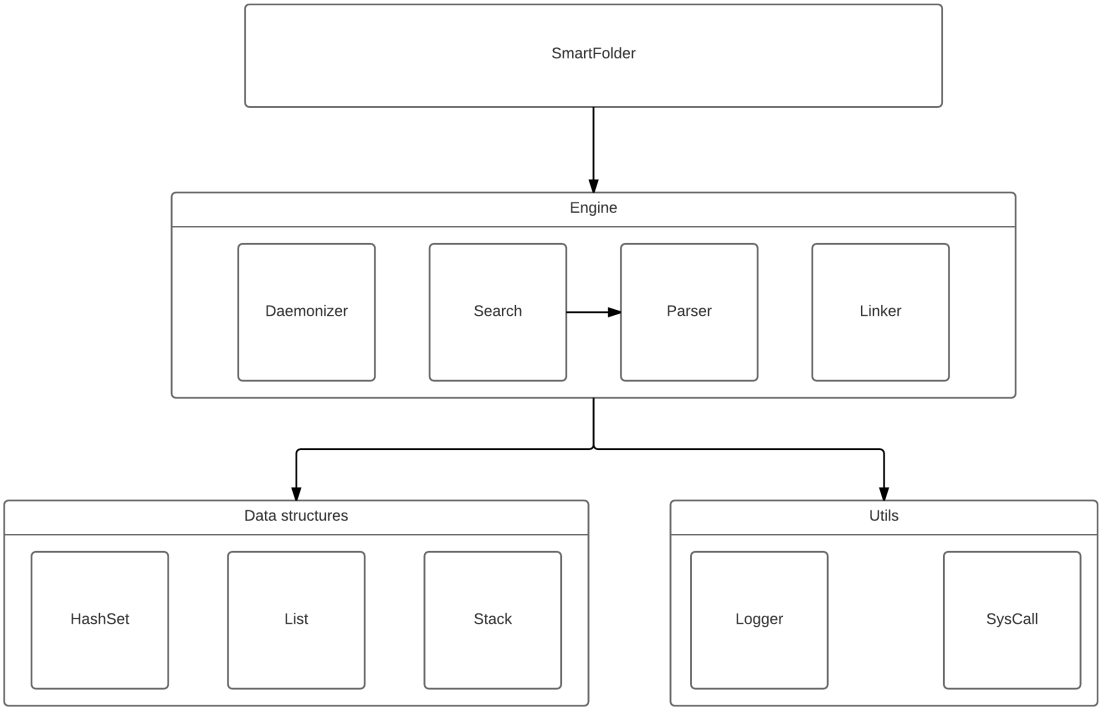
Maxime Lovino Thomas Ibanez

22 décembre 2016

# 1 Introduction

Hello world

# 2 Architecture




## 2.1 Data Structures

### 2.1.1 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.

```
#ifndef _LIST_H_
#define _LIST_H_

typedef struct listElement_st {
    char* data;
    struct listElement_st* next;
} ListElement;

typedef struct list_st {
    ListElement* head;
    int size;
} List;

List* initList();
void insert(List* l, char* element);
void removeIndex(List* l, int idx);
void removeObject(List* l, char* element);
int searchInList(List* l, char* element);
char* get(List* l, int idx);
List* listUnion(List* l1, List* l2);
```

```
List* listIntersect(List* l1, List* l2);
List* listXOR(List* l1, List* l2);
List* listComplement(List* l1, List* l2);
void deleteList(List* l);

#endif /* end of include guard: _LIST_H_ */
```

### 2.1.2 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 $\Theta(n)$).

```
#ifndef _HASH_SET_
#define _HASH_SET_
#include "Logger.h"

typedef struct HashSet {
  char** table;
  int size;
  int filled;
} HashSet;


void initSet(HashSet* table, int size);
void expand(HashSet** table);
void put(HashSet** table, char* filePath);
void removeFromSet(HashSet* table, char* filePath);
int contains(HashSet* table, char* filePath);
int searchInSet(HashSet* table, char* filePath);
int hash(char* text);
void deleteSet(HashSet** table);
void dumpSet(HashSet* table);

#endif /* end of include guard: _HASH_SET_ */
```

### 2.1.3 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)

```
#ifndef _STACK_H_
#define _STACK_H_

typedef struct stackElement_st {
    struct stackElement_st* next;
    void* value;
} stackElement;

typedef struct stack_st {
    stackElement* top;
    int size;
} Stack;

Stack* initStack();
void push(Stack* s, void* element);
void* pop(Stack* s);
```

```c
int isEmpty(Stack* s);
void deleteStack(Stack* s);

#endif /* end of include guard: _STACK_H_ */
```

## 2.2 Utils

### 2.2.1 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 : INFO, un message d'information, par exemple un dump de hashtable ou des informations sur les fichiers trouvés
- 1 : WARNING, un message de warning, par exemple une erreur dans notre programme durant son execution mais qui n'empêche pas le programme de continuer
- 2 : 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.

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.

```c
#ifndef _LOGGER_H_
#define _LOGGER_H_
#ifndef LOG_LEVEL
#define LOG_LEVEL 0
#endif

#include <stdio.h>
#include <stdarg.h>

void logMessage(int level, const char* format, ...);

#endif /* end of include guard: _LOGGER_H_ */
```

### 2.2.2 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.

## 2.3 Engine

### 2.3.1 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.

```c
#ifndef _SEARCH_H_
#define _SEARCH_H_
#include "List.h"
#include <sys/stat.h>
```

```
typedef enum {NAME, SIZE_SMALLER, SIZE_EQUAL, SIZE_BIGGER, STATUS_DATE_B,
STATUS_DATE_E, STATUS_DATE_A, MODIF_DATE_B, MODIF_DATE_E, MODIF_DATE_A, USAGE_DATE_B,
USAGE_DATE_E, USAGE_DATE_A, OWNER, GROUP, MODE} searchType;

/**
 * Searches recursivly through directories beneath a certain directory
 * @param  rootDir      The directory to start from
 * @param  type       The type of search to perform
 * @param  searchArg  The argument for the search
 * @return              A list of matching files
 */
List* searchDirectory(char* rootDir, searchType type, void* searchArg);

/**
 * Compares 2 timespecs
 * @param  t1 the first timespec
 * @param  t2 the second timespec
 * @return    -1 if t1 < t2, 0 if t1 = t2 and 1 if t1 > t2
 */
int timeCompare(struct timespec* t1, struct timespec* t2);
#endif /* end of include guard: _SEARCH_H_ */
```

### 2.3.2 Parser

Ici nous avons la grande partie du programme, avec principalement la fonction `int evaluateAndSearch(char** expressi` 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.

```
#ifndef _PARSER_H_
#define _PARSER_H_
#define SECONDS_IN_YEAR 3.154e7
#define SECONDS_IN_MONTH 2.628e6
#define SECONDS_IN_DAY 86400
#define FILE_PATH_REGEX "((\\.|\\.\\.)?\\/)?(([A-Z]|[a-z]|[0-9]| |-|_|'|\\.)+\\/?)+"
#include <time.h>
#include <pwd.h>
#include <grp.h>
#include <string.h>
#include <regex.h>
#include <stdlib.h>
#include <stdio.h>
#include "Search.h"
#include "HashSet.h"
#include "List.h"
#include "Stack.h"
#include "Logger.h"
#include <ctype.h>

int isValidPath(char* path);
int getUID(char* userName);
int getGID(char* groupName);
struct timespec* getTimeSpec(char* date);

/**
```

```
 * Converts a search word to a searchType
 * @param  param the search word
 * @param  arg   the parameter linked to the search word
 * @return       the corresponding searchType or -1 if no match exists
 */
searchType getSearchType(char* param, char* arg);

/**
 * Checks if a word is a boolean operator
 * @param  word the word to check
 * @return      1 if it is a boolean operator, 0 otherwise
 */
int isBooleanOp(char* word);

/**
 * Checks if a searchType has proper argument linked to it
 * @param  st  The seach type
 * @param  arg The argument
 * @return     1 if the seach is valid, 0 otherwise
 */
int isValidSearch(searchType st, char* arg);

/**
 * Evalutes an expression and searches for matching files
 * @param  expression The expression to search with
 * @param  exprLen    The length of the expression
 * @param  folder     The folder to search from
 * @param  result     A map to put the result of the search
 * @return            0 if the search was ok, 1 otherwise
 */
int evaluateAndSearch(char** expression, int exprLen, char* folder, HashSet** result);

/**
 * Removes the + or - before a search argument
 * @param  st  searchType
 * @param  arg argument to trim
 * @return     Trimed argument
 */
char* trimArgument(searchType st, char* arg);

#endif /* end of include guard: _PARSER_H_ */
```

### 2.3.3 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(1).txt`)

### 2.3.4 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.

## 2.4 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.

# 3 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>`

## 3.1 Syntaxe de recherche

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

- `--name <name>`
- `--size [-+]<size>`
- `--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`
- `OR`
- `XOR`
- `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`