Programmation système
SmartFolder

David Gonzalez - Claudio Sousa

26 janvier 2017

1 Introduction

Ce TP de deuxième année en programmation système consiste à implémenter un programme similaire au SmartFolder sur MacOSX.

Le SmartFolder sur MacOSX recherche sur le disque des fichiers correspondant à un/des critères et, pour chacun des fichiers trouvés, le programme crée un lien symbolique dans un dossier specifié.

1.1 Spécification fonctionnelle

Ce programme possède deux modes de fonctionnement.

1.1.1 Mode recherche

C'est le mode par défaut et simule le SmartFolder sur MacOSX.

Dans ce mode, le programme tourne en arrière-plan et maintient dans le dossier de destination une liste de liens vers les fichier trouvés correspondant aux critères de recherche choisis. Cette liste est dynamique et mise à jour si des nouveaux fichiers répondent aux critères de recherche ou si, au contraire, des fichiers ne répondent plus à ces critères.

La recherche de fichiers est récursive et suit les liens symboliques.
Par ailleurs, les fichiers en double ne doivent pas apparaître et les fichiers portant le même nom doivent être renommés intelligemment.

Ce mode prend 3 paramètres :
- *<dir_name>* : chemin (de destination) où stocker les liens ;
- *<search_path>* : chemin de recherche ;
- *<expression>* : critères de sélection.

<dir_name> et *<search_path>* sont de simples chemins vers des dossiers.
<expression> correspond à une liste de critères dont l'interface est un sous-ensemble de celle de *find*. Son usage complet est décrit en annexe, section Expression.

1.1.2 Mode stop

Le mode *stop* permet d'arrêter une recherche en cours.

Ce mode prend en paramètre :
-d <dir_name> : termine le SmatFolder pour le chemin de destination specifié.

2 Development

2.1 Architecture

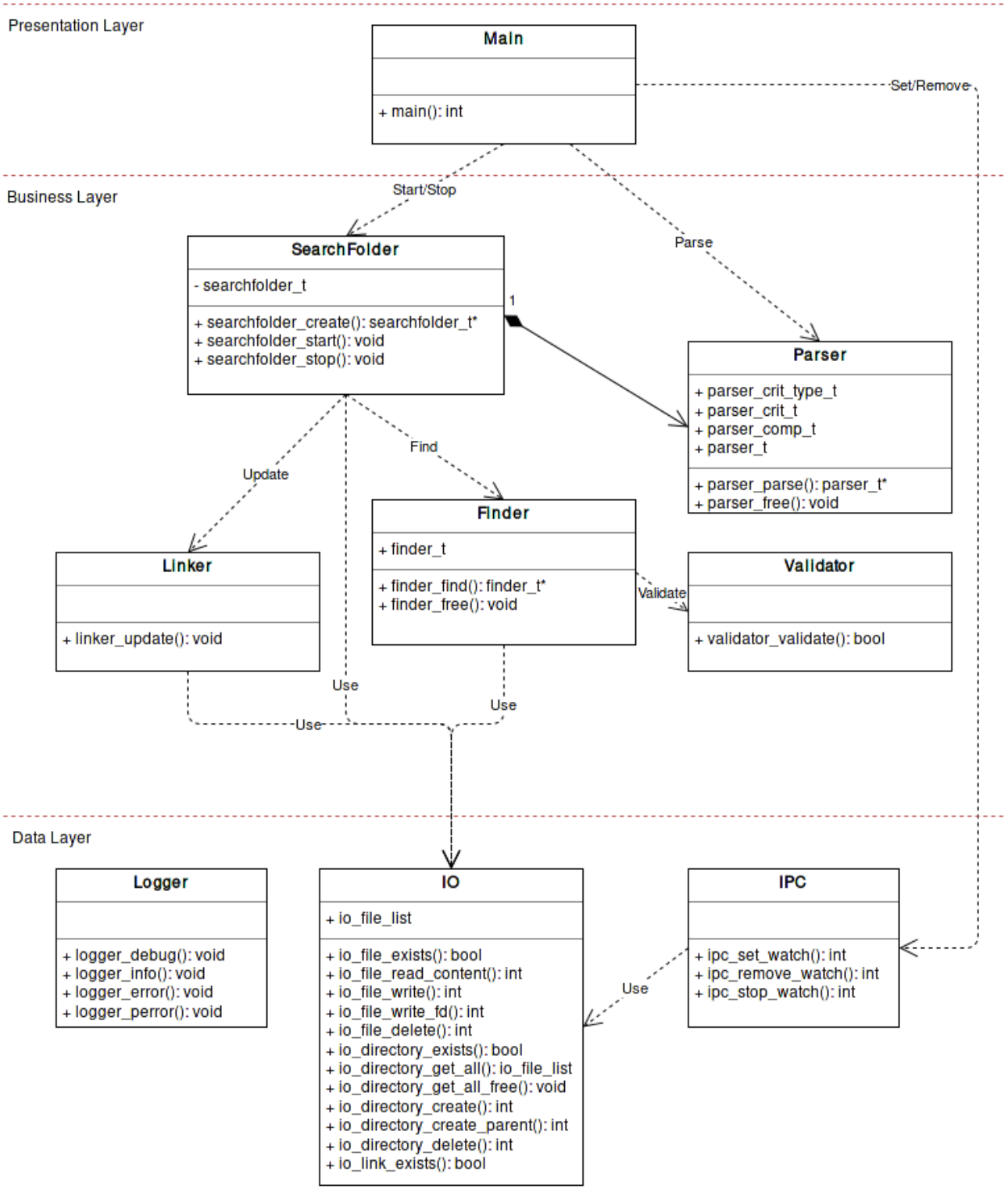

FIGURE 1 – Architecture du SmartFolder

2.1.1 Main

Le programme principal a pour rôle de vérifier les arguments et de sélectionner le bon mode de fonctionnement.
Dans le mode *recherche*, il a pour tâche de :

- met le processus en arrière-plan ;
- demande au module *Parser* de traiter l'expression ;
- initialise le module *IPC* ;
- initialise et lance le module *SmartFolder*.

Dans le mode *stop*, son seul rôle est de signaler l'arrêt à l'autre instance (voir IPC).

2.1.2 SmartFolder

SmartFolder est le module principal qui va orchestrer la recherche et la mise à jour du dossier de destination.

Lorsque lancé, il va continuellement utiliser le module *Finder* pour rechercher les fichiers correspondant au critère, puis donner la liste des fichiers retournée au module *Linker* pour qu'il mette à jour le répertoire de destination.
A noter qu'entre chaque recherche, il y a une pause de quelques secondes.

A l'arrêt, il est chargé de détruire le répertoire de destination et de libérer ses ressources.

2.1.3 Parser

Parser est le module qui transforme l'expression spécifiée dans la ligne de commande (critères de recherche) en une structure interne utilisable par le module *Validator*.

La complexité des différents opérateurs logiques possibles (priorité, parenthèses, opérateur unaire) doit être connue uniquement de ce module. La structure retournée doit pouvoir être traitée simplement par le module *Validator*.

Suggestion d'implémentation : utiliser l'algorithme de *shunting-yard*[1] pour le traitement de l'expression.

2.1.4 Validator

Ce module vérifie si un fichier est valide selon l'expression créée par le module *Parser*.

2.1.5 Finder

Ce module est responsable de produire la liste de tous les fichiers du dossier de recherche respectant les criteres de recherche.
La vérification des fichiers contre les critères est déleguée au module *Validator*.

Lors du processus de recherche effectué au sein de ce module, un parcours d'arborescence est effectué récursivement et les liens symboliques sont suivis.
Des boucles deviennent alors possibles. Ce module doit donc gérer ce cas spécifique et prévenir les boucles infinies ainsi que le traitement des fichiers et dossiers dupliqués.

Possibilité d'implémentation : utilisation d'une table d'hashage contenant les numéros d'inode des fichiers et répertoires déjà parcourus.

1. `https://en.wikipedia.org/wiki/Shunting-yard_algorithm` (par Edsger Dijkstra)

2.1.6 Linker

Le module *Linker* a pour rôle de mettre à jour le dossier de destination à l'aide d'une liste de fichiers passée en paramètre.

Pour chaque fichier, il crée un lien si celui-ci n'existe pas. Les liens qui ne sont plus valides sont effacés.

2.1.7 IPC

Ce module a pour but de répondre au problème du mode *stop*. En effet, lorsqu'une instance de *SmartFolder* souhaite arrêter une autre instance en cours d'exécution, il faut d'une manière ou d'une autre permettre une communication simple entre les deux processus afin qu'une instance puisse signaler un arrêt à une autre instance.

Le moyen de communication choisi est les signaux POSIX, le signal *SIGTERM* pour être plus précis. Une instance qui veut donc signaler l'arrêt doit envoyer le signal cité au processus concerné.

Afin de pouvoir lancer un signal, il faut que le PID du processus cible soit connu. Pour cela, le PID d'une instance en mode *recherche* est stocké dans un fichier dans le répertoire utilisateur.
Comme une instance de *SmartFolder* est unique par dossier de destination, ce fichier se nommera d'après le chemin de ce répertoire.

2.1.8 IO

Le but de ce module est d'offrir une interface simple aux appels systèmes et de centraliser la gestion des erreurs.

2.1.9 Logger

Ce module centralise l'affichage des logs de débogage.

2.2 Flux général d'exécution

Le diagramme ci-dessous décrit le flux général de l'application :

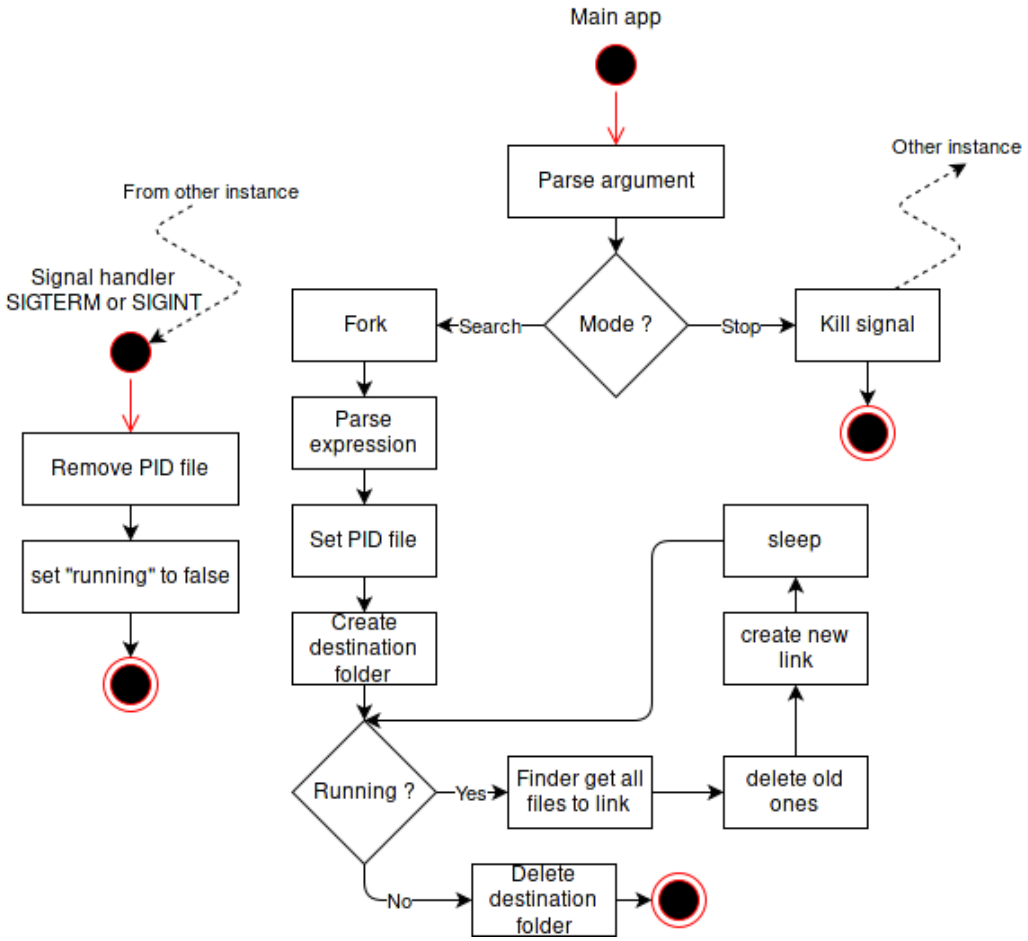

FIGURE 2 – Flow général d'exécution

L'application commence par décoder le premier argument pour savoir dans quel mode il doit être exécuté.

En mode *recherche*, il commence par se mettre en arrière-plan. Puis, il donne le reste des argument au Parser d'expression pour la validation des fichiers. Ensuite, il écrit son PID dans le fichier et crée le dossier de destination donnée en argument pour pouvoir commencer la recherche.

La recherche tourne indéfiniment tant qu'un signal d'arrêt (SIGTERM ou SIGINT) n'est pas reçu. Cette recherche commence par parcourir récursivement l'ensemble des fichiers dans le dossier de recherche donnée en argument. Puis, avec cette liste, le programme met à jour les liens dans le dossier de destination en commençant par purger les liens qui ne figurent plus dans la liste, puis crée ceux qui y sont nouveaux. La recherche se termine par une attente de quelques secondes et recommence.

En mode *stop*, il se contente de lire le PID dans le fichier créé par l'instance cible et lui envoie un signal de terminaison (SIGTERM). Ceci provoque l'exécution d'un *handler* sur l'instance concernée qui place un indicateur pour que la boucle de recherche s'arrête.

3 Implémentation

La documentation d'implémentation a été générée à l'aide de Doxygen en version HTML et se trouve à l'addresse : `https://hepia-projects.gitlab.io/smart-folder/`.

Tous les modules possèdent une description complète en début de fichier. Toutes les fonctions publiques ont été documentées dans le fichier d'header, alors que les fonctions privées ont été documentées dans le fichier d'implémentation.

Le diagramme d'architecture (figure 1) donne toutes les fonctions et structures publiques de chaque module.

Le code source, testes unitaires et source du rapport se trouve à l'adresse `https://gitlab.com/hepia-projects/smart-folder/tree/master`.

3.1 Testes unitaires

Le module *Parser* a été entièrement testé à l'aide de CuTest. Ce module fut choisit en raison de sa complexité. Les tests unitaires se trouve dans le dossier *tests* et peut être exécuté avec *make run*.

L'image ci-dessous donne la sortie des tests unitaires du module *Parser* :

```
Test: parse empty                                                    [   OK   ]
Test: parse incomplete exp                                           [   OK   ]
Test: parse incorrect exp                                            [   OK   ]
Test: parse name                                                     [   OK   ]
Test: parse name contains                                            [   OK   ]
Test: parse group                                                    [   OK   ]
Test: parse wrong group                                              [   OK   ]
Test: parse user                                                     [   OK   ]
Test: parse wrong user                                               [   OK   ]
Test: parse permission                                               [   OK   ]
Test: parse perm. contains                                           [   OK   ]
Test: parse wrong permission                                         [   OK   ]
Test: parse size                                                     [   OK   ]
Test: parse wrong size                                               [   OK   ]
Test: parse atime                                                    [   OK   ]
Test: parse ctime                                                    [   OK   ]
Test: parse mtime                                                    [   OK   ]
Test: parse wrong time                                               [   OK   ]
Test: parse operators                                                [   OK   ]
Test: parse parenthesis                                              [   OK   ]
Test: parse wrong parenthesis                                        [   OK   ]
Test: -not -name test                                                [   OK   ]
Test: -name test -or -size 20                                        [   OK   ]
Test: -not -name test -or -size 20                                   [   OK   ]
Test: -name test -or -not -size 20                                   [   OK   ]
Test: -not ( -name test -or -size 20 )                               [   OK   ]
Test: -name test -size 20                                            [   OK   ]
Test: -name test -not -size 20                                       [   OK   ]
Test: -name test -and -size 20                                       [   OK   ]
Test: -user root -and -size 20 -or -name test                        [   OK   ]
Test: -user root -and ( -size 20 -or -name test )                    [   OK   ]
Test: -name test -or -size 20 -and -user root                        [   OK   ]
Test: ( -name test -or -size 20 ) -and -user root                    [   OK   ]
Test: -group root -and -size 20 -or -not -user root -or -perm 777    [   OK   ]
Test: -group root -not -size 20 -or -user root -perm 777             [   OK   ]
Test: -group root -not ( -size 20 -or -user root ) -perm 777         [   OK   ]

Summary:
  SUCCESS: All unit tests have passed.
```

FIGURE 3 – Sortie des tests unitaires du module *Parser*

4 Annexe

4.1 Expression

L'interface est semblable à celle de *find*[2]. Voici la liste des options prises en charge :

Critères :

- *-name* : fname
  - Le nom de fichier est exactement fname
  - Exemple, les fichiers només "todo.txt" : *-name todo.txt*
- *-name* : -fname
  - Le nom de fichier contient fname.
  - Exemple, les fichiers contenant ".txt" : *-name -.txt*
- *-group* : gname
  - Le fichier appartient au group gname
  - Exemple, les fichiers appartenant au group *root* : *-group root*
- *-user* : uname
  - Le fichier est possédé par utilisateur uname
  - Exemple, les fichiers possédés par l'utilisateur *claudio* : *-user claudio*
- *-perm* : perm
  - Les permissions sont exactement perm
  - Exemple, les fichiers ayant exactement les permissions 644 : *-perm 644*
- *-perm* : -perm
  - Le fichier a au moins les permissions perm
  - Exemple, les fichiers dont l'utilisateur peut lire et executer : *-perm -500*
- *-size* : [+-]size[GMkc]
  - Le fichier a la taille size.
  - Le suffix indique l'unité utilisée (insensible à la casse) :
    - *'c'* : bytes (par défaut si non précisé)
    - *'k'* : Kilobytes
    - *'M'* : Megabytes
    - *'G'* : Gigabytes
  - Le préfixe indique la comparaison utilisée :
    - *'+'* : supérieur à
    - *'-'* : inférieur à
    - *''* : exactement
  - Exemples :
    - *-size 200c* : fichiers de taille 200 bytes
    - *-size -30k* : fichiers de taille inférieure à 30KB
    - *-size +2M* : fichiers de taille supérieure à 2MB
- *-atime* : [+-]time[dmhs]
  - Le fichier fut accédé depuis time. La référence de temps est le moment présent.
  - Le suffix indique l'unité utilisée (insensible à la casse) :
    - *'s'* : secondes (par défaut si non précisé)
    - *'m'* : minutes
    - *'h'* : heures
    - *'d'* : jours
  - Le préfixe *[+-]* indique la comparaison utilisée, comme pour le critère -size ci-dessus.
  - Exemples :
    - *-atime +5m* : fichiers accédés depuis plus de 5 minutes
    - *-atime -1d* : fichiers accédés dans les dernières 24 heures
- *-ctime* : [+-]time[md]
  - Le statut du fichier fut changé depuis time.

2. `https://linux.die.net/man/1/find`

Pour plus détails, voir l'explication pour le critère -atime

-mtime : [+-]time[md]

Le fichier fut changé depuis time.

Pour plus détails, voir l'explication pour le critère -atime

Les differents critères énumérés ci-dessus peuvent être combinés avec les opérateurs listés ci-dessous, énumérés dans l'ordre de précédence décroissante :

Opérateurs :

(critère) : force la précédence de l'expression entre parenthèses

-not : applique le *NOT* logique au critère de droite

-and : applique le *AND* logique entre les critères à gauche et à droite de l'opérateur

-or : applique le *OR* logique entre les critères à gauche et à droite de l'opérateur

4.2 Changelog

La liste ci-dessous décrit l'ensemble des changements apportés à ce document d'architecture.

- mise à jour du diagramme pour l'ajout de :
  - IO :
    - structure : *io_file_list* ;
    - fonction : *io_file_write_fd* ;
    - fonction : *io_directory_get_all* ;
    - fonction : *io_directory_create_parent* ;
    - fonction : *io_directory_delete* ;
    - fonction : *io_link_exists*.
  - Logger :
    - fonction : *logger_debug* ;
    - fonction : *logger_info* ;
    - fonction : *logger_error* ;
    - fonction : *logger_perror*.
  - Parser :
    - énumération : *parser_crit_type_t* ;
    - énumération : *parser_crit_t* ;
    - énumération : *parser_comp_t* ;
    - fonction : *parser_free*.
  - SmartFolder : renommé en SearchFolder.
- ajout d'une section *Flux général d'exécution* ;
- ajout d'une section *Implémentation* ;
- Expression :
  - les suffixes de taille (*size*) sont insensibles à la casse.
  - si le suffixe de taille (*size*) n'est pas précisé, alors l'unité par défaut est l'octet (byte).
  - ajout de deux suffixe de temps (*atime*, *ctime*, *mtime*) : 's' pour seconde et 'h' pour heures.
  - les suffixes de temps (*atime*, *ctime*, *mtime*) sont insensibles à la casse.
  - si le suffixe de temps (*atime*, *ctime*, *mtime*) n'est pas précisé, alors l'unité par défaut est la seconde.