HES-SO – HEPIA - SECTION ITI

Mini-projet « search folder »

Programmation Système

Etienne Guignard, Hassine Rmiza

Année 2016-2017
19/01/2017

Table des matières

1. Introduction

Dans le cadre du cours de programmation système, il nous a été demandé de mettre en place une gestion de répertoires virtuels de recherche de fichiers similaires à *Smart Folders* sous *MacOS/Search* ou *Folders* sous *GNU/Linux*.

2. Description du projet

Dans cette section, nous allons vous décrire les différentes fonctionnalités mises en place dans le programme, les structure de données ainsi que les commandes acceptées par le programmes.

2.1 Main

La fonction main récupère les arguments que l'utilisateur a entré en ligne de commande. Puis, on vérifie juste que le nombre d'arguments est supérieur au nombre minimum d'arguments qu'il est nécessaire d'introduire pour que le programme fonctionne. Puis, on passe les arguments correspondant aux options au parseur afin de définir ce que l'utilisateur veut rechercher.

2.2 Parser

Pour que l'utilisateur puisse entrer des arguments afin d'effectuer une recherche sur un fichier, nous avons mis en place une fonctionnalité que sépare les différents arguments et les vérifient. Le parseur vérifie les informations suivantes :

- Le nombre d'arguments
- Présence des dossiers *search* et *link*
- Chemin absolu ou relatif

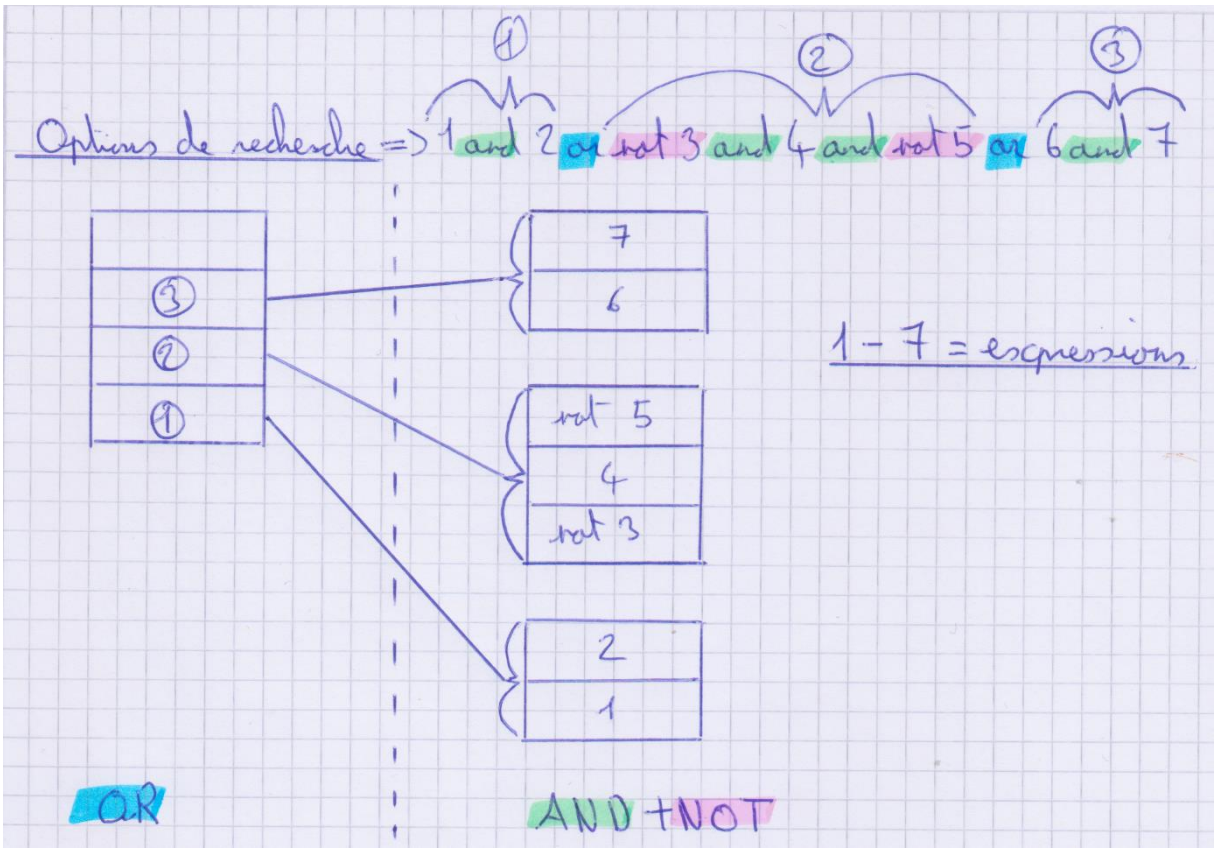
Si le parseur détecte une erreur, l'utilisateur est notifié et le programme se termine. Dans le cas contraire, deux options existent : soit le programme s'arrête et effectue l'option de stopper le daemon, soit il démarre le daemon pour commencer la recherche des fichiers. S'il n'y a pas d'erreurs, le parseur combine les arguments dans une structure, puis passe la main au daemon.

Par manque de temps nous n'avons pas implémenter le mécanisme permettant de combiner plusieurs expressions avec des opérateurs de type booléen. Toutefois, nous avons quand même réfléchi à la démarche que nous aurions suivie pour mettre en place ce mécanisme. Voici les opérateurs que nous aurions voulu mettre en place :

- *AND*
- *OR*
- *NOT*

Pour parser avec des expressions booléennes, nous avons adopté la méthode suivante. On parcourt la chaine de caractères contenant les options. Dès que l'on a une expression, on stocke le résultat dans une liste. Dans le cas contraire, on vérifie si c'est un *OR*, *AND* ou *NOT*. Si c'est un *OR*, on prend le résultat de l'expression suivante et on la stocke dans une nouvelle liste. Si c'est un *AND*, on prend le résultat de l'expression suivante et on ajoute le résultat à la liste courante et ceci jusqu'à que l'on rencontre un nouveau *OR*. Pour le *NOT*, on traite avec le résultat de l'expression associée puisque c'est un opérateur unaire. Une fois toute la chaine parcourue, on peut donner le résultat des recherches au

daemon. Puis, à partir de ces résultats, on s'occupe de créer des liens symboliques pour les fichiers recherchés.



2.3 Daemon

La partie daemon est la fonctionnalité qui permettra au processus de tourner en tâche de fond afin que la recherche sur un dossier ne s'arrête pas jusqu'à ce que l'utilisateur le demande ou que la machine soit éteinte. Pour ce faire, après avoir récupéré les arguments, on effectue un fork qui va créer un processus fils. Le processus père crée donc un fichier *daemon.dm*, qui va stocker le *PID* du processus fils ainsi que le nom du dossier où l'on effectue la recherche. Puis, on quitte le processus père ce que veut dire que le processus fils sera en tâche de fond mais on ne le verra pas. Pour stopper la recherche, le nouveau processus père récupère dans le fichier le *PID* correspondant au dossier de recherche et quitte le processus fils.

2.4 Files

Cette fonctionnalité regroupe toute la partie recherche des fichiers par rapport à ce que l'utilisateur à préciser en argument. Au démarrage du programme, on fait une recherche récursive dans le dossier donné. Puis, on compare chaque fichier trouvé avec l'expression. Si elle correspond, on ajoute un lien symbolique dans le dossier donné, puisque l'on ne veut pas effectuer une recherche en continue, ce que surchargerait le *CPU*. Nous avons décidé d'employer la librairie *inotify*, qui nous notifie quand un fichier a bien été créé dans le dossier de recherche.

2.5 linkedlist

Pour stocker les attributs des fichiers durant le processus de recherche, nous avons employé une liste chainée, car elle est plus simple à manipuler qu'un simple tableau. De plus, l'allocation de mémoire se fait dynamiquement ce qui permet d'insérer une grande quantité de données.

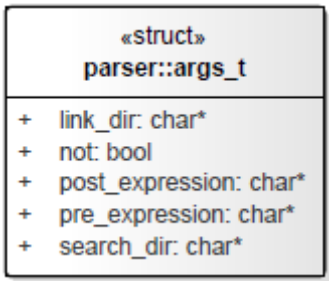
2.6 Function

Dans cette partie, plusieurs fonctionnalités ont été regroupées, car elles n'avaient pas un lien direct avec les fonctionnalités mentionnées précédemment.

2.7 Structure de données

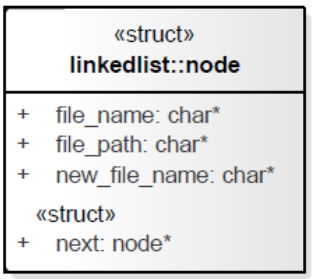
La structure *args* contient les arguments après avoir été parsée. Voici dans l'ordre la description de chaque variable :

- *Link_dir* : nom du répertoire où sont créés les liens
- *Not* : permet de savoir si l'utilisateur veut l'inverse de ce qu'il demande
- *Pre_expression* : le type de recherche, par exemple : par nom, date etc…
- *Post_expression* : la valeur associée au type de recherche
- *Search_dir* : nom du répertoire où faire la recherche



La structure *node* contient les informations relatives au fichier pendant la recherche. Voici dans l'ordre la description de chaque variable :

- *File_name* : le nom du ficher
- *File_path* : le chemin du fichier
- *File_new_name* : le nom du fichier en cas de doublon



2.8 Commandes

Les commandes suivantes peuvent être exécutées avec notre programme. Pour la commande qui assure la recherche des permissions, il y a une petite particularité. Le caractère "-" indique que si l'une des permissions pour l'utilisateur, le groupe, ou tous les utilisateurs est posée, alors le fichier comparé sera ajouté. Quant au caractère "/", il indique que si l'un des bits est setté pour l'utilisateur, le groupe, ou tous les utilisateurs, alors le fichier comparé sera ajouté.

```
[link_dir] [search_dir] [-not]          [-name]  [text]
[link_dir] [search_dir] [-not]          [-size]  [<+-> <number of byte> <KMGT>]
[link_dir] [search_dir] [-not] [-date]  [-atime] [<+-> <number of day>]
[link_dir] [search_dir] [-not] [-date]  [-amin]  [<+-> <number of min>]
[link_dir] [search_dir] [-not] [-date]  [-ctime] [<+-> <number of day>]
[link_dir] [search_dir] [-not] [-date]  [-cmin]  [<+-> <number of min>]
[link_dir] [search_dir] [-not] [-date]  [-mtime] [<+-> <number of day>]
[link_dir] [search_dir] [-not] [-date]  [-mmin]  [<+-> <number of min>]
[link_dir] [search_dir] [-not] [-user]          [name]
[link_dir] [search_dir] [-not] [-group]         [name]
[link_dir] [search_dir] [-not] [-perm]          [<-/> <number>]
[-d] [link_dir]
```

3. Tests

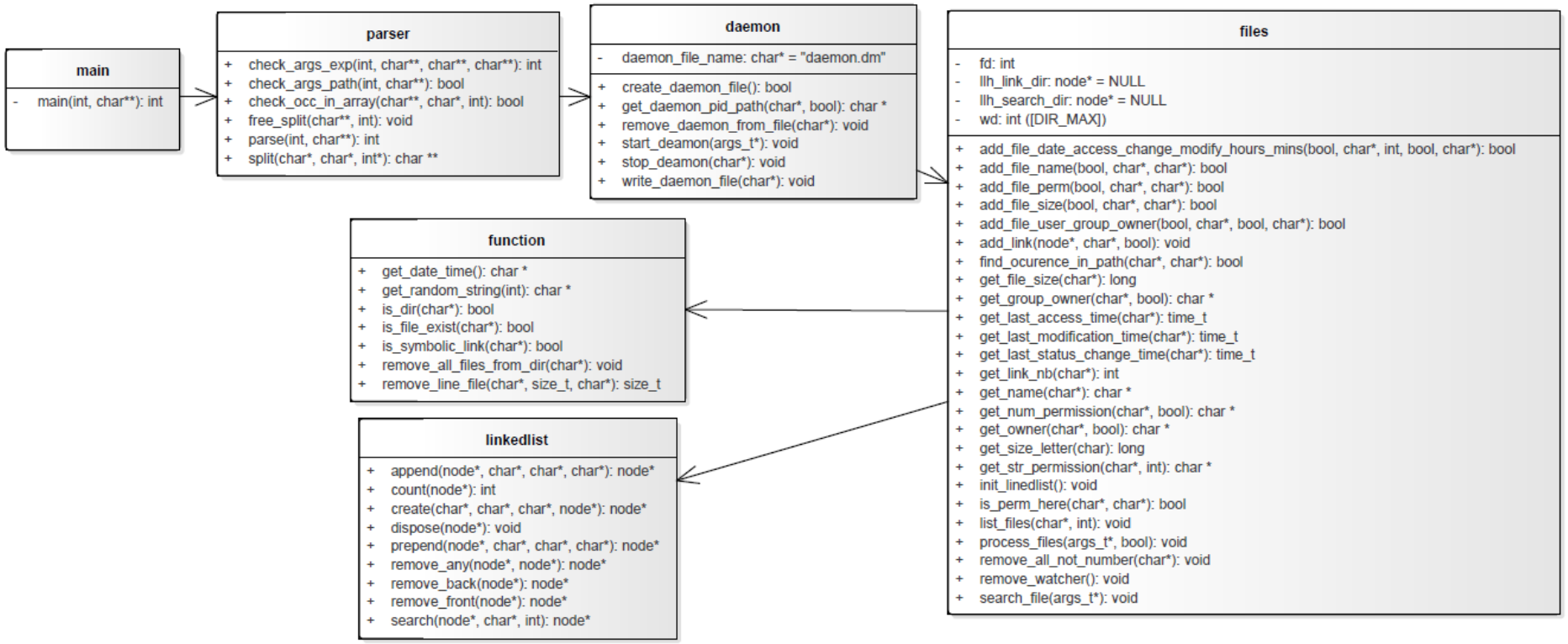
Pour garantir le bon fonctionnement de notre programme, nous avons effectué des tests. Nous avons procédé de la manière suivante. Avec un script *bash* nous avons exécuté toutes les commande de base de notre programme, puis le résultat a été comparé avec le résultat de la commande *find* ainsi que les arguments correspondant. Nous avons aussi effectué un test avec le programme *Valgrind* afin de savoir si notre programme avait des fuites de mémoire.

N°	Description	Résultat	Résolution
01	Création dossier qui contiendra les liens	OK	-
02	Création du daemon + ficher	OK	-
03	Suppression du daemon + ligne dans le ficher + dossier lien + ficher lien	OK	-
04	Détection de boucles sur les liens symboliques	OK	-
05	Suivi des liens symboliques	OK	-
06	Recherche par nom	OK	-
07	Recherche par taille <>=	OK	-
08	Recherche par date <>=	OK	-
09	Recherche par propriétaire	OK	-
10	Recherche par droits, identique ou un seul posé	OK	-
11	Ajout d'un fichier en cours d'exécution	OK	-
12	Fuite de mémoire avec *Valgrind*	OK	-

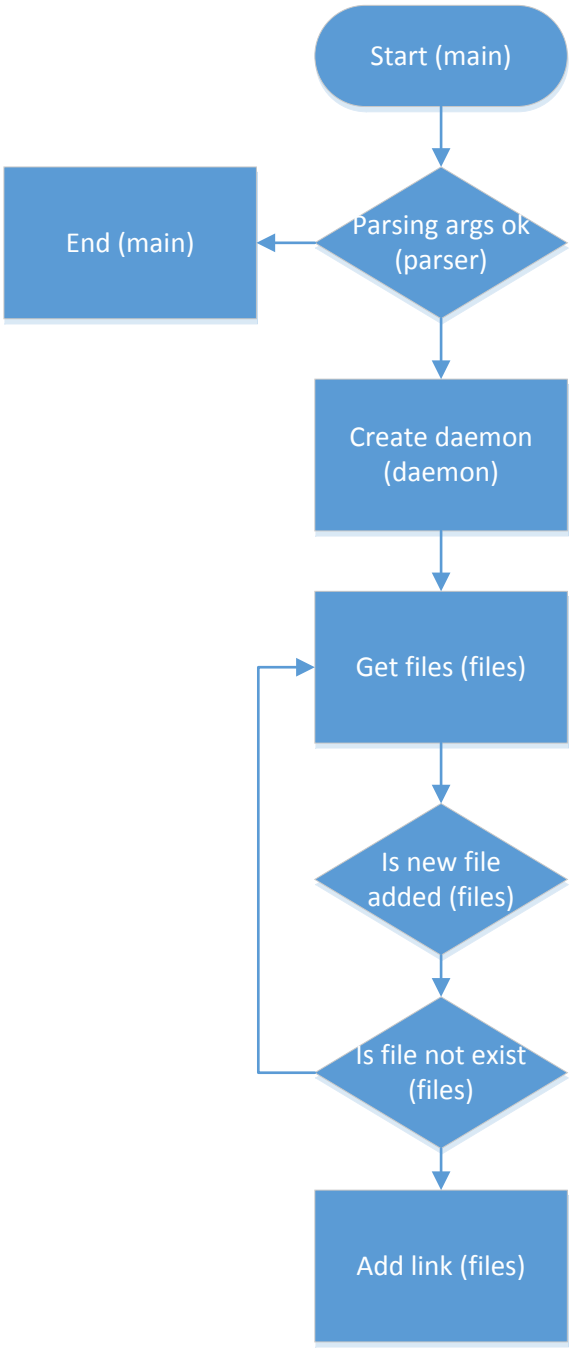
4. Schémas

Cette section est dédiée au schéma de principe ainsi que le diagramme de class en *UML*.

4.1 UML



4.2 Principe de fonctionnement



5. Conclusion

Notre projet est fonctionnel et permet d'effectuer le travail demandé. Aucun bug n'a été observé lors des tests que nous avons effectués, mais tous les cas possibles n'ont sûrement pas été testés, car étant nombreux. Nous avons changé notre implémentation entre le rendu intermédiaire et celui-ci, car nous jugions plus pertinent d'effectuer le travail d'une autre façon.

6. Sources

- http://www.linux-france.org/article/memo/node126.html
- http://www.tutonics.com/2012/12/find-files-based-on-their-permissions.html
- http://www.thegeekstuff.com/2012/02/c-daemon-process/
- http://stackoverflow.com/questions/17954432/creating-a-daemon-in-linux
- http://www.thegeekstuff.com/2010/04/inotify-c-program-example/
- http://pubs.opengroup.org/onlinepubs/9699919799/functions/nftw.html
- http://www.t1shopper.com/tools/calculate/
- http://www.geeksforgeeks.org/generic-linked-list-in-c-2/
- https://www.programiz.com/c-programming/c-dynamic-memory-allocation
- http://stackoverflow.com/questions/20716785/how-do-i-delete-a-specific-line-from-text-file-in-c
- http://stackoverflow.com/questions/15758678/read-a-line-from-text-file-and-delete-it