mickael.hoerdt@hesge.ch 2016/2017

Programmation Système/Systèmes d'exploitation
Mini-projet

Un répertoire virtuel de recherche de fichiers

1. Cahier des charges

L'objectif de ce projet est d'implémenter en langage C et sur la base d'appels système Unix, une idée similaire aux Smart Folders[1] sous MacOS/Search Folders[2] sous GNU/Linux. Le but pour l'utilisateur est de pouvoir créer un répertoire « virtuel » d'un type spécial contenant le résultat d'une requête de recherche en tant que liens symboliques vers les fichiers originaux du système de fichier.

Les liens symboliques sont différents de liens directs : à la différence des liens directs, les liens symboliques encodent simplement le chemin d'accès du fichier vers lequel ils pointent. Le contenu du fichier pointé par un lien symbolique peut très bien ne pas exister ou bien exister sur un périphérique de stockage qui n'est pas encore accessible à ce moment là.

1. Votre interface aux répertoires de recherche virtuels doit avoir la forme d'un outil en ligne de commande dont l'utilisation sera la suivante :

   ```
   searchfolder <dir_name> <search_path> [expression]
   searchfolder -d <dir_name>
   ```

2. La première commande créera un nouveau répertoire appelé du nom donné par le paramètre `dir_name`. Ce répertoire contiendra les liens symboliques vers les fichiers situés sous le chemin `search_path` et correspondant à la recherche indiquée dans l'expression `expression`. La deuxième commande désactivera une recherche associée au répertoire `dir_name` exécutée en arrière plan puis effacera le répertoire `dir_name` associé.

3. L'expression donnée en paramètre à la première commande aura un format similaire à celui accepté par la commande Unix `find(1)`, en plus simple. Au minimum, votre outil devra pouvoir interpréter les types d'expression suivants :

   a) Nom du fichier correspondant exactement à une chaîne de caractères ou bien contenant une sous-chaîne de caractères ascii.
   b) Taille de fichier plus grande, plus petite, ou égale à une valeur donnée.
   c) Date de création/modification/utilisation du fichier plus ancienne, plus récente, ou égale à une valeur donnée.
   d) Propriétaire ou groupe propriétaire égal ou différent d'une valeur donnée.
   e) Droits d'accès du fichier correspondant à une valeur exacte donnée, ou contenant ou moins l'un des droits posés donnée ou tous les droits posés demandés.
   f) Une expression booléenne combinant a) b) c) d) e) en implémentant par exemple un mécanisme similaire aux « opérateurs » de `find(1)`.

1 http://support.apple.com/kb/PH14047
2 http://en.wikipedia.org/wiki/Virtual_folder#Windows

4. Une fois instanciée, la recherche devra être active en arrière-plan pour mettre à jour le répertoire virtuel dans le cas où de nouveaux fichiers correspondants à la recherche apparaîtraient sous le système de fichier. Par exemple l'utilisateur **ne** doit **pas** relancer la recherche à chaque fois qu'il désire connaître le nom de tous les fichiers de type image dans son répertoire home.

   La recherche instanciée dans le répertoire `search_path` devra être **récursive**, c'est à dire qu'elle contiendra les résultats contenus dans le répertoire `search_path` ainsi que ses sous-répertoires (et ainsi de suite). La recherche instanciée devra pouvoir suivre des liens symboliques, détecter les boucles de répertoires et les éviter. Il doit être possible de créer un `searchfolder` dont `dir_name` est aussi un `searchfolder`.

   La recherche instanciée ne contient que des fichiers, elle de contient pas de répertoire. Si deux fichiers ont le même nom, l'outil doit le détecter et y remédier par une méthode de votre choix.

5. Exemple de résultat d'exécution :

```
$ searchfolder /tmp /home/hoerdtm/grosfichiers -size +10M
$ ls -l /home/hoerdtm/grosfichiers
vacances_2014.mp4 -> /tmp/video/film_de_vacances_2014.mp4
transparents_cours_os.pdf -> /tmp/transp_cours_os.pdf
```

   Après exécution de cette commande, si de nouveaux fichiers d'une taille supérieure à 10 méga-octets devaient apparaître sous le répertoire `/tmp`, il devraient automatiquement apparaître dans le répertoire `/home/hoerdtm/grosfichiers`. De même, si les fichiers trouvés sont effacés dans `/tmp`, les liens vers les fichiers dans `/home/hoerdtm/grosfichiers` doivent l'être aussi.

2. Aide

Les fonctions suivantes, toutes documentées dans les pages de manuel des sections indiquées entre parenthèses, peuvent vous être utiles :

- `strcmp(3)` pour trouver des symboles dans une chaîne et `strtok(3)` pour séparer une chaîne de caractères en tokens.
- L'appel système `fork(2)` pour créer un processus en arrière plan.
- `getcwd(3)` et `chdir(2)` pour obtenir le répertoire de travail courant ou bien le changer.
- `opendir(3)` pour ouvrir un répertoire et `readdir(3)` pour en lire une entrée. Vous pouvez utiliser ces fonctions pour trouver dans quel répertoire se trouve le programme à exécuter. Voir aussi l'appel système `stat(2)` qui permet d'obtenir des informations d'attributs sur les fichiers (taille, propriétaire, uid, gid, etc...)
- `rmdir(2)` et `unlink(2)` pour effacer un répertoire ou un lien.

3. Modalités de rendu

Le projet est à réaliser par groupe de 2 ou 3 personnes.

Un rapport de maximum **5 pages** au format **pdf** décrivant les différents composants de votre implémentation ainsi que leur interdépendances devra être rendu par mail avant le **23/12/2016**.

Un rapport de maximum **5 pages** au format **pdf** documentant votre implémentation devra être rendue par mail au plus tard le **26/1/2017** avant 23h59.

Votre code C doit être compilable via un fichier `Makefile` sous GNU/Linux et `gcc`.
Une archive du code au format **<nom1_nom2>.zip** (ou **<nom1_nom2_nom3>.zip** pour un groupe de 3 personnes) devra être rendue par mail au plus tard le **26/1/2017** avant 23h59.

Un projet réalisé par un groupe de 3 personnes devra contenir dans l'archive rendue une suite de tests unitaires permettant de tester le bon fonctionnement de chaque composants implémentés pour ce projet à travers un fichier Makefile et/ou des scripts bash.