Projet de Programmation Système
Système de gestion de fichiers avec tags

Auteurs:

CASTELBOU Louis
CHEKROUN Hiba-Asmaa
MIRET Blanche

Sommaire

Présentation du système

Le programme répond aux exigences de l'implémentation d'un système de gestion de fichier avec tag. L'installation se fait en exécutant le script *install.sh*; toutes les fonctionnalités sont accessibles via la commande *tag* comme décrit dans la documentation (voir README). La copie d'un fichier copie également ses tags mais les deux versions du fichier évoluent ensuite indépendamment. Concernant l'utilisation, nous avons opté pour une séparation explicite entre les commandes de gestion du tagset (la liste des tags organisée par une hiérarchie) et celles d'association des tags à des fichiers. Nous distinguons donc l'opération de création de tag, qui l'ajoute à la liste des tags et l'insère dans la hiérarchie, et l'opération d'association de tag à un fichier : il est nécessaire de créer un tag avant de pouvoir l'associer à un fichier. Par ailleurs, il n'est pas possible d'avoir deux tags du même nom, et l'ordre des options et des arguments n'est pas interchangeable.

Les fonctionnalités et commandes sont décrites dans le README. Une documentation est fournie avec une simili page de manuel, affichée avec `%tag`, et une page d'aide affichée avec `%tag --help`.

Architecture

Description des scripts et modules

Installation et désinstallation

- **install.sh :** script d'installation.
- **uninstall.sh** : script de désinstallation.

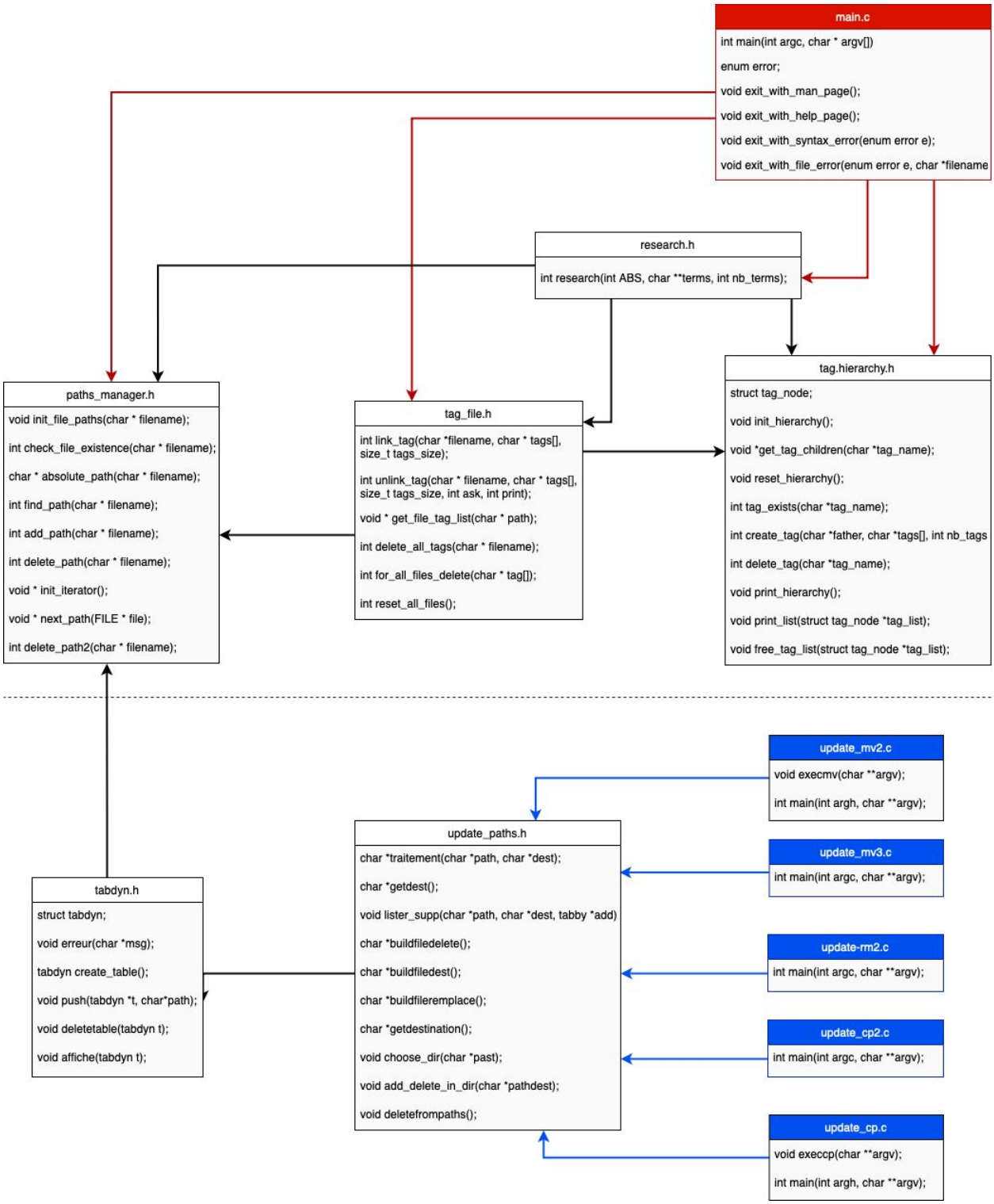
Programme principal

- **main.c** : centralise les actions des différents modules selon l'option et les arguments entrés par l'utilisateur en ligne de commande.
- **tag_hierarchy.c** : gère la lecture et l'écriture de la liste des tags et sa hiérarchie dans le fichier *tag_hierarchy*.
- **tag_file.c** : gère l'attribution des tags aux fichiers et leur suppression.
- **paths_manager.c** : gère le fichier contenant les chemins des fichiers taggés.
- **research.c** : gère la recherche de fichiers taggés par combinaison de tags.

Alias de commandes

- **tabdyn.c** : implémentation d'un tableau dynamique
- **update_paths.c** : gère le suivi des fichiers taggés
- **update-mv*.c** : gère le suivi des fichiers avant et après l'exécution de mv.
- **update-rm2.c** : gère le suivi des fichiers après l'exécution de rm.
- **update-cp*.c** : gère le suivi des fichiers avant et après l'exécution de cp.
- **cpt.sh :** script qui permet de relier la commande cp au système de tag (alias).
- **mvt.sh :** script qui permet de relier la commande mv au système de tag (alias).
- **rmta.sh :** script qui permet de relier la commande rm au système de tag (alias).

Diagramme


main.c
int main(int argc, char * argv[])
enum error;
void exit_with_man_page();
void exit_with_help_page();
void exit_with_syntax_error(enum error e);
void exit_with_file_error(enum error e, char *filename
research.h
int research(int ABS, char **terms, int nb_terms);
paths_manager.h
void init_file_paths(char * filename);
int check_file_existence(char * filename);
char * absolute_path(char * filename);
int find_path(char * filename);
int add_path(char * filename);
int delete_path(char * filename);
void * init_iterator();
void * next_path(FILE * file);
int delete_path2(char * filename);
tag_file.h
int link_tag(char *filename, char * tags[], size_t tags_size);
int unlink_tag(char * filename, char * tags[], size_t tags_size, int ask, int print);
void * get_file_tag_list(char * path);
int delete_all_tags(char * filename);
int for_all_files_delete(char * tag[]);
int reset_all_files();
tag.hierarchy.h
struct tag_node;
void init_hierarchy();
void *get_tag_children(char *tag_name);
void reset_hierarchy();
int tag_exists(char *tag_name);
int create_tag(char *father, char *tags[], int nb_tags
int delete_tag(char *tag_name);
void print_hierarchy();
void print_list(struct tag_node *tag_list);
void free_tag_list(struct tag_node *tag_list);
update_mv2.c
void execmv(char **argv);
int main(int argh, char **argv);
update_paths.h
char *traitement(char *path, char *dest);
char *getdest();
void lister_supp(char *path, char *dest, tabby *add)
char *buildfiledelete();
char *buildfiledest();
char *buildfileremplace();
char *getdestination();
void choose_dir(char *past);
void add_delete_in_dir(char *pathdest);
void deletefrompaths();
update_mv3.c
int main(int argc, char **argv);
update-rm2.c
int main(int argc, char **argv);
update_cp2.c
int main(int argc, char **argv);
update_cp.c
void execcp(char **argv);
int main(int argh, char **argv);
tabdyn.h
struct tabdyn;
void erreur(char *msg);
tabdyn create_table();
void push(tabdyn *t, char*path);
void deletetable(tabdyn t);
void affiche(tabdyn t);

Structures et techniques employées

Association des tags aux fichiers

> tag_file.c

Les tags sont stockés comme des attributs étendus (*extended attributes*) via les fonctions fournies par la librairie *xattr*. Les attributs étendus sont des extensions des attributs normaux, ces derniers sont associés à tous les inodes du système. Ils sont représentés sous la forme d'une paire constituée de nom,valeur. Dans notre projet, nous avons utilisé les attributs étendus *user* de la manière suivante: user.tags = "couleur/film/policier/rose/", ici le nom qu'on a choisi est "tags", et la valeur est "couleur/film/policier/rose/", il s'agit d'un ensemble de tags, chaque tag est séparé par un slash, ce symbole est également rajouté après le dernier tag. Nous avons choisi d'utiliser des attributs car ils sont conservés de manière permanente lorsqu'on utilise les commandes mv et cp.

Installation

> install.sh

Le script d'installation install.sh a la tâche d'installer tous les fichiers permettant l'utilisation du projet : tout d'abord il installe les paquets xattr et libglib2.0-dev, sans lesquels le programme ne peut rien faire. Il crée ensuite un répertoire .tag (/.tag) dans lequel il va placer tous les fichiers qui peuvent intervenir lors de l'exécution du programme puis enchaîne en concevant les fichiers qui vont stocker les chemins des fichiers taggés et la hiérarchie des tags, exécute le make et compile ce qui va remplacer certaines commandes (mv, rm, cp), puis déplace tout ce beau monde dans /.tag. Il va achever sa tâche en rendant les exécutables contenus dans /.tag accessibles partout dans le système de fichier et en relançant le terminal dans lequel l'installation fut lancée. On appelle ensuite la commande alias sur les scripts correspondant (alias mv="mvt.sh", alias cp="cpt.sh", alias rm="rmta.sh"). Comme le répertoire est à la racine le suivi des fichiers est commun aux différents utilisateurs, cependant pour utiliser le système un utilisateur tierce doit lancer install.sh pour que les exécutables de /.tag soient ajoutés au PATH et pour que les alias sur les commandes soient faits.

Liste et hiérarchie des tags

> tag_hierarchy.c
> tag_hierarchy

La liste des tags et leur hiérarchie est conservée dans le fichier *tag_hierarchy* sur lequel sont énumérés les tags sous la forme : *nom_père - nom_tag*.

À la demande de création d'un tag, le fichier est parcouru et chaque tag déjà présent lu un par un. Si un de ceux-ci porte le même nom que celui à créer, ou si à la fin de la lecture le nom du père précisé par l'utilisateur n'a pas été trouvé, la commande s'arrête. Sinon, la nouvelle paire est inscrite à la fin, *nom_père* devenant "root" si aucune hiérarchie n'a été précisée et que le tag est placé à la racine du tagset. Ainsi, dans la liste des tags inscrits sur le fichier *tag_hierarchy,* un tag arrive systématiquement avant ses fils, mais pas forcément immédiatement avant, ce qui a de l'importance à la lecture.

Chaque lecture et écriture de tags sur ce fichier se fait via la structure *struct tag_t* qui contient deux champs pour *nom_père* et *nom_tag,* ainsi que des pointeurs vers les structures représentant les frères du tags et ses enfants. La fonction *build_tree()* permet de construire au fil de la lecture du fichier *tag_hierarchy* l'arbre représentant toute la hiérarchie des tags avec cette structure, en s'aidant d'une hashmap composée en même temps. Cette dernière facilite la recherche d'un tag dans l'arbre : ses clés sont les noms des tags, et les valeurs l'adresse des *struct tag_t* dans l'arbre de la hiérarchie. Ainsi, à la lecture d'une nouvelle paire *nom_père - nom_tag*, une nouvelle structure est créée, son adresse est ajoutée à la hashmap pour la clé "nom_tag" ; elle est ensuite reliée à la structure représentant son père dont l'adresse est retrouvée via la hashmap.

La hashmap est également mise à profit quand il est nécessaire d'accéder à un tag pour retrouver ses enfants. C'est le cas à la demande de la suppression d'un tag : dans ce cas l'arbre de hiérarchie est construit, la sous-hiérarchie ayant ce tag pour racine est identifiée, supprimée de l'arbre, et l'arbre intégralement réécrit dans le fichier *tag_hierarchy*. C'est aussi le cas au moment de la recherche par combinaison de tag, opération décrite plus bas.

Suivi des fichiers taggés

> paths.txt
> mvt.sh
> cpt.sh
> rmta.sh

Parcourir toute l'arborescence du système de fichier à chaque recherche de tag était une méthode qui ne nous semblait absolument pas viable d'un point de vue utilisateur car la vitesse de la recherche ne serait alors pas optimale. C'est pour cette raison que nous avons décidé de stocker les chemins absolus des fichiers taggés dans le fichier *paths.txt*. Avec cette solution il fallait faire en sorte que *paths.txt* soit toujours à jour, même quand on déplace, supprime ou copie un fichier. Nous avons donc créé des alias pour les commandes *mv*, *cp* ainsi que *rm* pour pallier ce problème. Nous parcourons uniquement les répertoires qui vont être déplacés et le répertoire de destination sur 1 niveau pour gérer les potentielles copies. À chaque fois nous regardons tous les chemins de path.txt : si un chemin n'est plus accessible nous le supprimons du fichier.

Recherche de fichier par combinaison de tags

> research.c

Une combinaison énumère des noms de tags, précédés ou non par un indicateur d'exclusion. Pour correspondre à la recherche, un fichier doit être taggé par chacun des tags inclus, ou par l'un de leurs enfants, et ne doit être taggé par aucun des tags exclus ni par aucun de leurs enfants. Au lancement d'une recherche, un travail préliminaire est fait en constituant une liste de tags par terme de la recherche, chaque liste contenant le tag lui-même et tous ses enfants. Ces listes sont marquées par un entier 0 ou 1 précisant leur type : "inclus" ou "exclus". Comme vu plus haut, la liste des fichiers taggés est retenue et maintenue dans un fichier, permettant d'énumérer facilement chaque candidat à la recherche. L'algorithme de recherche est le suivant :

```
Pour chaque fichier F
|     Pour chaque liste L de tags
|     |     Pour chaque tag T du fichier :
|     |     |     Si L est du type "inclus" :
|     |     |     |     Si T appartient à L : passer à la liste suivante
|     |     |     Sinon :
|     |     |     |     Si T appartient à L : passer au fichier suivant
|     Afficher le chemin du fichier
```

Tests

Pour tester le système, des scripts sont fournis dans le répertoire *tests* :

- *test_all_commands.sh* : teste toutes les commandes de tag, notamment la partie recherche.

- *error_managing_test.sh* : vérifie la bonne réaction du programme face à des erreurs d'utilisation, arguments manquants ou erronés.

- *create_basic_test_environment.sh* : créé un tagset et des fichiesr taggés pour procéder à des tests libres.b