Projet "Systèmes"
2A TELECOM Nancy - 2022-2023

Rapport d'activités

Membres de l'équipe projet :
Quentin LENFANT, Damien SIMON

Professeur référent :
Maïwenn RACOUCHOT

Table des matières

1 Algorithme général

1.1 Parseur

La première étape du programme consiste à parser l'entrée standard afin de récupérer le paramètre de la commande `ftc` (que l'on appelera par la suite *source*), les différents flags et les paramètres qui sont associés. Cependant, les paramètres de certains flags doivent respecter des contraintes propres à ceux-ci. La problématique était donc de traiter l'entrée standard de façon systématique en tenant compte du format des paramètres spécifiques à chaque flag.

Pour répondre à ce problème, nous avons tout d'abord utilisé un tableau `opt_table` où chaque élément correspond à un flag. Ces flags sont eux-mêmes représentés par une structure `option` qui contient les attributs suivants :

— le nom du flag
— un booléen pour savoir si le flag est présent (activé) dans l'entrée standard
— la valeur du paramètre du flag
— un pointeur vers la fonction de vérification de la validité du paramètre (`check_flag_param()`, retourne un booléen)
— un pointeur vers la fonction de filtrage du flag (`flag_filter()`, retourne un booléen)

À noter que pour simplifier les traitements, la *source* est également un flag, activé par défaut.

Nous avons également appliqué le principe de *diviser pour régner* pour la vérification des paramètres des flags. Au travers d'une boucle for générique sur les éléments de `opt_table`, nous appelons pour chaque flag la fonction `check_flag_param()` correspondante et contrôlons le paramètre associé au flag (qu'il y en ait un ou non).

En appliquant ces stratégies et en considérant le fait que toutes les fonctions `check_flag_param()` ont la même signature et sont toutes écrites dans un fichier `check_param.c` séparé du parseur, l'algorithme de ce dernier peut ainsi traiter l'entrée standard de façon systématique et indépendamment des flags implémentés. Cela apporte également plus de flexibilité dans le développement de `ftc.c`.

1.2 Parcours récursif

La deuxième étape consiste ensuite à parcourir l'arborescence des fichiers depuis la `source` et de filtrer les résultats en fonction. Le sujet nous imposait de programmer un parcours récursif et la problématique était similaire au parseur : avoir un parcours qui filtre les résultats de façon systématique tout en tenant compte des spécificités des flags implémentés.

Nous avons alors mis en place un parcours récursif en profondeur. Certains fichiers sont cependant ignorés pour éviter des appels récursifs infinis : c'est le cas du fichier `"."` à la *source* (à ne pas confondre avec le chemin de la *source* qui est passé dans les filtres) et tous les fichiers `".."`. Tous les autres fichiers, répertoires ou non, sont considérés et les appels récursifs se font sur les répertoires. La fonction de parcours affiche également en sortie standard les fichiers qui après filtrage remplissent les contraintes indiquées dans l'entrée standard.

La phase de filtrage est caractérisée par la fonction `filter()`, appelée à chaque étape du parcours et renvoyant un booléen qui vaut 1 si le fichier en paramètre vérifie tous les filtres (comportement ET logique, par défaut) ou au moins un filtre (comportement OU logique, voir 2.3.3). On applique également durant cette phase le principe de *diviser pour régner* de façon analogue aux vérifications de paramètres des flags. Le filtrage de l'arborescence s'effectue flag par flag pour chaque fichier à l'aide des fonctions `flag_filter()`. Ces dernières ont également toutes la même signature et sont écrites dans un fichier `filters.c` séparé, ce qui, comme pour le parseur, rend le filtrage indépendant des flags implémentés. Cela permet ainsi un développement plus flexible du code de `ftc.c`.

2 Implémentation des fonctions liées aux flags

2.1 Mise en place des *regex*

La prise en charge des *regex* a été mise en place à l'aide de la librairie `regex.h`. Les *regex* ont été utilisés lorsque les flags devait les implémenter d'une part (voir les flags `-name`, `-dir` et `-ctc`), et lors de la phase de vérification des paramètres d'autre part.

Dans ce deuxième cas, l'idée était de vérifier le format à respecter par le paramètre du flag à l'aide d'une regex et cela était utilisé pour les flags `-size`, `-date`, `-perm` et `-threads`.

Par exemple, le flag `-size` devait commencer par un +, un - ou aucun signe, puis comporter au moins un chiffre entre 0 et 9 et enfin se terminer par une unité de taille (c, k, M ou G). La *regex* utilisée pour valider le paramètre du flag donné en entrée standard est `[+-]?[0-9]+[ckMG]`.

2.2 Flags requis

2.2.1 -test

Le flag `-test` a pour but d'afficher en sortie standard le paramètre d'un flag, valide ou non, sous la forme "`La valeur du flag -`*`nom_du_flag`* `est` *`valeur_du_flag`*".

Le cas du flag `-test` est traité à part dans le `main()` de `ftc.c` (lorsqu'il est activé, on parse bien l'entrée standard, mais on n'effectue pas le parcours/filtrage de l'arborescence). Dans le cas où il n'y aurait qu'un seul flag dans l'entrée standard en plus de `-test`, ce dernier affiche le paramètre de ce flag. S'il y a plusieurs flags, `-test` affichera la valeur du premier flag activé rencontré dans le parcours de `opt_table`. À noter cependant que la `source`, bien que considérée comme un flag particulier et présent dans `opt_table`, n'est pas prise en compte par `-test` (pas de "`La valeur du flag -source est` *`source`*" en sortie standard). Ainsi, en l'absence de flag supplémentaire, `-test` n'affiche rien.

2.2.2 -name

Le flag `-name` effectue un filtrage par nom de fichier. La valeur du flag est soit un nom de fichier complet, soit une *regex*. Il faut cependant noter qu'en l'absence des caractères `.*` au début et/ou à la fin du paramètre du flag, les noms des fichiers recherchés doivent correspondre exactement à la valeur du flag. La difficulté réside donc ici dans la gestion de cette recherche exacte.

L'implémentation choisie consiste à transformer dans tous les cas le paramètre du flag en *regex*. La contrainte d'exactitude est alors remplie avec l'ajout artificiel à cette *regex* des caractères `^` (début d'un mot) et `$` (fin d'un mot) qui force ainsi l'utilisation de `.*`. Il suffit ensuite de comparer cette *regex* avec le nom du fichier étudié.

2.2.3 -dir

Le flag `-dir` effectue un filtrage par répertoire. Le flag possède ou non un paramètre qui est un nom de répertoire (nom complet ou *regex*)

Le fonctionnement et l'implémentation de cette fonctionnalité sont analogues à ceux de `-name`. On effectue simplement une vérification supplémentaire pour savoir si le fichier étudié est bien un répertoire. Pour cela, on utilise la variable `st_mode` de `stat()`.

2.2.4 -size

Le flag `-size` effectue un filtrage par rapport à la taille du fichier. Suivant les cas, le fichier doit être soit plus grand, soit plus petit, soit de la même taille que celle donnée par le paramètre du flag

Pour cela, nous comparons la valeur du flag avec la taille du fichier étudié, obtenue avec la variable `st_size` de `stat()`.

2.2.5 -date

Le flag `-date` effectue un filtrage par rapport à la date de dernier accès au fichier. La durée depuis le dernier accès est soit inférieure ou égale (cas par défaut), soit supérieur ou égal au paramètre du flag.

Nous avons alors comparé la valeur du flag avec la différence entre l'heure au moment de l'exécution de `ftc` (fonction `time()`) et la date de dernier accès au fichier obtenue avec la variable `st_mtime` de `stat()`. (En théorie, nous aurions dû utiliser la variable `st_atime` de `stat()`, mais pour une raison inconnue, c'est bien `st_mtime` qui correspond à la donnée recherchée.)

2.2.6 -mime

Le flag `-mime` effectue un filtrage par rapport au type mime d'un fichier. Il prend en paramètre un type mime de la forme *type* ou *type/sous-type*.

Pour implémenter cette fonctionnalité, nous avons utilisé la librairie `MegaMimes`. Dans la fonction `check_mime_param()`, après un prétraitement sur le format du paramètre du flag (s'il est de la forme *type*, on le transforme en *type/**), on appelle la fonction `getMegaMimeExtensions()` qui nous renvoie la liste des extensions associées au type mime, sinon `null` (et dans ce cas, le paramètre du flag est invalide). Pour le filtrage, la valeur du flag est mise sous la forme `type/`. On note n la longueur de ce paramètre formaté. On compare alors les n premiers caractères du type mime du fichier étudié (obtenu avec `getMegaMimeType()`) avec ce paramètre formaté.

2.2.7 -ctc

Le flag `-ctc` effectue un filtrage par rapport au contenu des fichiers. Il a pour but de vérifier si une chaîne de caractère (symbolisée ou non par une *regex*) est présente dans le fichier étudié.

Pour ce faire, nous utilisons une chaîne de caractère `buffer` de longueur égale à la taille du fichier en paramètre, et dans laquelle on stocke à chaque itération le contenu d'une ligne du fichier. On effectue donc la recherche de motif (chaîne de caractère ou `regex` en paramètre de `-ctc`) sur ce buffer.

2.3 Flags facultatifs

2.3.1 -color

Une des fonctionnalités supplémentaires proposées était de mettre en place un système de couleur pour l'affichage des résultats.

Nous avons pour cela utilisé les *escape characters* en définissant au préalable les couleurs utilisées dans le fichier `ftc`. Les répertoires sont affichés en jaune, les autres fichiers en bleu, et les messages d'erreurs sur l'invalidité des paramètres des flags en rouge. Les autres messages d'erreurs restent cependant dans la couleur par défaut.

2.3.2 -perm

Nous avons choisi de mettre en place le filtrage des fichiers en fonction des permissions. Ce flag prend en paramètre les permissions recherchées sous format octal.

Pour l'implémentation, la fonction `check_perm_param()` utilise une *regex* pour vérifier que le paramètre possède exactement 3 chiffres compris entre 0 et 7. La fonction de filtrage utilise quant à elle la variable `st_mode` obtenue avec la fonction `stat()` et transtypée en objet de type `__uintmax_t` (ou `unsigned long`). Il suffit alors de récupérer les 3 derniers chiffres de cette variable et de les comparer au paramètre du flag.

2.3.3 -ou

Par défaut, chaque fichier parcouru par `ftc` doit répondre à tous les critères imposés par les flags invoqués dans l'entrée standard (fonctionnement assimilable à un ET logique sur les flags). Le but de cette option est donc d'afficher le fichier en sortie standard s'il vérifie au moins un flag (OU logique).

Cette fonctionnalité a été implémentée dans la fonction `filter()` de `ftc.c` qui prend en paramètre un fichier. Chaque filtre renvoyant un booléen, nous effectuons donc un ET logique d'une part et un OU logique d'autre part sur les booléens renvoyés par chaque filtre. La fonction `filter()` renvoie le résultat du OU logique si le flag est activé par l'utilisateur, le résultat du ET logique sinon.

2.3.4 -threads et -link

Pour ces deux flags, seules les fonctions de vérification des paramètres `check_link_param()` (qui est en réalité la fonction `check_no_param()`) et `check_threads_param()` ont été implémentées.

3 Volume horaire du travail effectué

	Quentin LENFANT	Damien SIMON
Conception	5h	15h
Implémentation	36h	28h
Tests	12h	10h
Rédaction du rapport	4h	1h
TOTAL	57h	54h