Rapport de Projet RS 2018/2019

Samer Abou Jaoude
Ishara Chan-Tung

Encadrants : Nicolas Schnepf
Lucas Nussbaum

Table des matières

1 Introduction

Notre projet est de créer la commande shell rsfind. Cette commande est un outil de recherche multi-critères de fichiers.
Rsfind permet d'afficher tous les fichiers d'un répertoire et des répertoires sous-jacents. Elle offre aussi la possibilité de faire une recherche plus fine avec les différentes options qu'elle propose comme la recherche de texte dans un fichier ou bien l'affichage de permissions.

2 Travail effectué

2.1 Bibliothèques utilisées

— De la librarie standard usuelle : stdio,stdlib,stdbool,stddef,unistd,string,fcntl,fnmatch,dirent
— Du système : stat,wait,dlfcn
— Pour l'affichage de détails (-l) : pwd,grp,time
— Pour la recherche d'image (-i) : libmagic
— Pour la recherche d'expressions régulières PERL (-T) : libprce2

2.2 Choix de conception

2.2.1 Généralités

Nous avons choisi d'organiser le dépôt de la manière suivant :
— **./ :** contient rsfind, Makefile, .gitlab-ci.yml...
— **./src :** contient l'ensemble des sources (*.c, *.h)
— **./obj :** contient les fichiers objets générés par la compilation
— **./test :** contient des fichiers sur lesquels on réalise des tests
— **./doc :** contient la documentation HTML générée par doxygen
— **./test_output :** contient le résultat de nos propres tests
Make se charge de créer test_output et obj s'ils n'existent pas.
Afin de faciliter l'implémentation, nous avons séparé le code source en plusieurs modules repartis sur des fichiers différents. Cela nous permettait de travailler en parallèle et d'éviter les conflits de merge, tout en clarifiant la structure du programme.
La compilation de chaque module se fait séparément puis l'assemblage se fait avec le linker.

L'algorithme de rsfind se décompose en plusieurs étapes :
— parsing des options
— recherche récursive des fichiers répondant aux critères (-i, -t, –name, ...)
— application itérative du traitement sur ces fichiers (–print, –exec, -l)

2.2.2 Structure du programme

Le programme étant de taille relativement petite, nous nous sommes permis d'utiliser deux variables globales pour faciliter la communication entre les différents modules.
— **args_t myArgs** les arguments passés à rsfind, parsés.*(structures.h)*
— **dynamic_array filesFound** contient l'ensemble des fichiers répondant aux critères de recherche (-i, -t, –name, ...).*(structures.h)*
— **struct magic** contient des pointeurs vers les fonctions de libmagic.*(image.h)*

Différents modules :

- **dynamic_array.h :** classe tableau dynamique
- **opt_parser.h :** parse les options de rsfind et remplit la structure *args_t myArgs*
- **cmd_parser.h :** parse la commande passée au –exec
- **cmd_exec.h :** execute la commande du –exec sur un fichier
- **display.h :** affichage simple ou détaillé(ls -l) d'un fichier
- **image.h :** traitements relatifs à libmagic
- **text_matcher.h :** fonctions relatives à la correspondance de chaînes de caractères (-T,-t,–name,–ename)
- **process.h :** détermine quelle action appliquer sur un fichier (–print, -l, –exec, -p)
- **rsfind.h :** listing récursif des fichiers du répertoir, les stock dans *dynamic_array filesFound*
- **sugar.h :** Macros pour alléger la syntaxe
- **structures.h :** définition des types de base et des variables globales
- **main.c :** s'occupe d'ordonnancer le tout

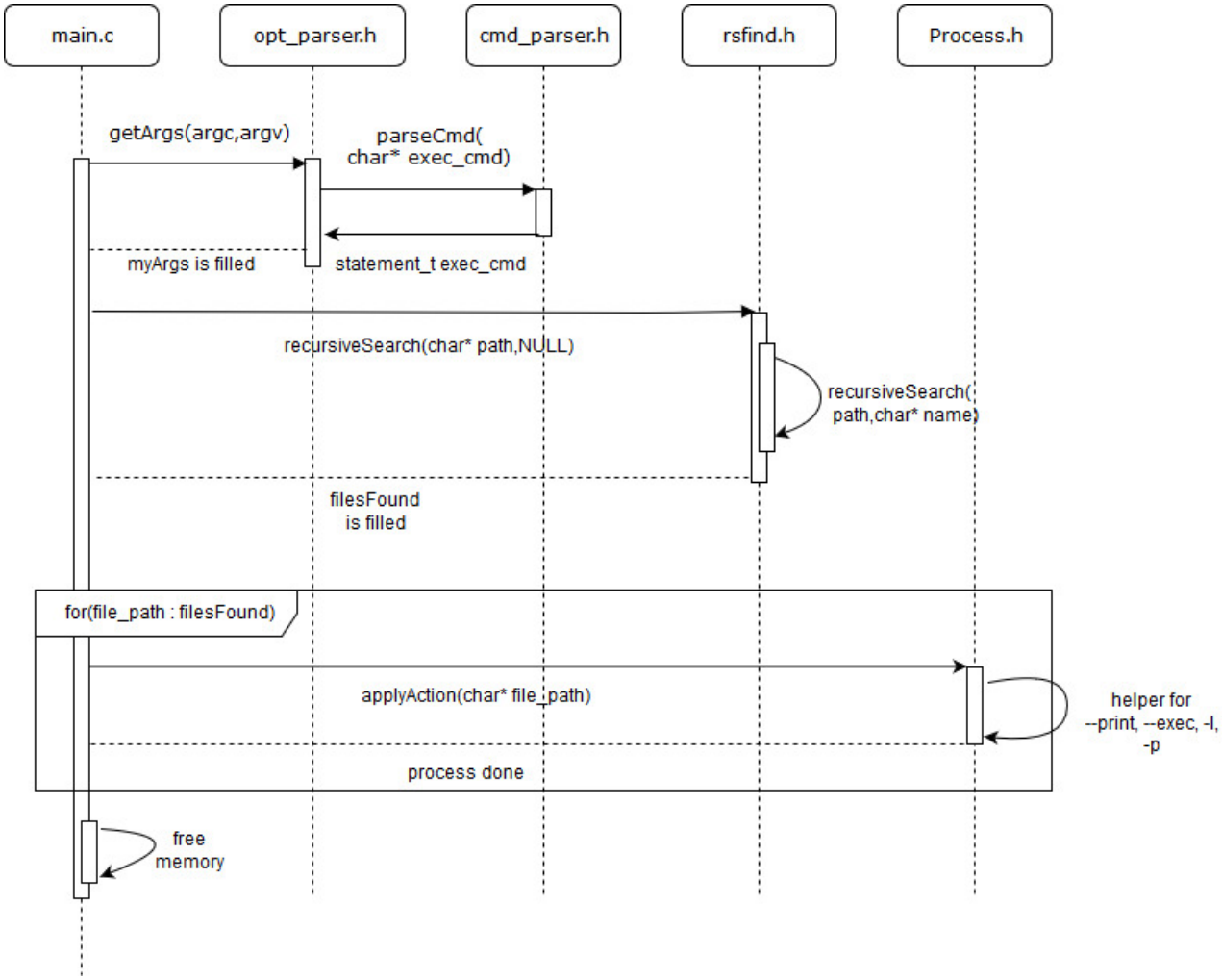

FIGURE 1 – Diagramme de séquence de l'exécution de rsfind

2.3 Options à implémenter

2.3.1 Etape 1 : Analyse des options de la ligne de commande

Pour analyser la liste d'options passées en argument de rsfind nous avons utilisé la fonction **get_opt_long**. Cette fonction reconnaît les paramètres qu'on lui fait connaître par avance qu'ils soient longs ou courts. De plus si un argument ne commence pas par un tiret alors il est mis en fin de liste donc on peut accéder facilement à cet argument qui est le chemin.

2.3.2 Étape 2 : listing du contenu d'un répertoire (–name et –print)

Pour parcourir les fichiers d'un répertoire, nous avons utilisé la libraire "dirent.h" avec les fonctions **opendir,readdir**, on parcoure tous les fichiers "enfants" au répertoire courant et on les affiche avec les informations de la structure dirent (notamment le nom du fichier).

Pour l'option -name, avant d'afficher un fichier on regarde si le nom du fichier est le même que le nom passé en argument avec l'option –name.

2.3.3 Étape 3 : listing des sous-répertoires

Pour parcourir les sous-répertoires, on a tout d'abord fait un "childPath" pour les fichiers dans le répertoire courant. On concatène le path du "père", c'est-à-dire le chemin du répertoire courant et le nom du fichier "fils" obtenu grâce à dirent. La libraire stat nous permet de savoir si un fichier est un répertoire ou non avec *S_ISDIR*. Si le fichier est un dossier, on le parcourt récursivement et on l'affiche si c'est un fichier normal, on l'affiche normalement.

2.3.4 Étape 4 : listing détaillé (-l)

Pour obtenir les informations intéressantes sur les fichiers nous avons utilisé les librairies stat, time, grp et pwd.
En faisant les affichages nous avons été très rigoureux en ce qui concerne les espacements et l'ordre des composants.

2.3.5 Étape 5 : recherche de texte (-t)

Afin de savoir si un texte est contenu dans un fichier, nous avons ouvert celui-ci (on ne l'ouvre que si c'est un fichier et pas un répertoire) et récupéré son contenu dans une chaîne de caractère. Puis avec la fonction **strstr** nous avons comparé les deux chaînes de caractères. Si la fonction ne renvoie pas NULL c'est qu'une occurrence de la chaîne de caractères voulue a été trouvée.

2.3.6 Étape 6 : recherche d'images (-i)

Pour la recherche d'images nous avons utilisé la librairie **libmagic**. Par celle-ci on peut récupérer une chaîne de caractère "magic_pipe" qui commence par le type du fichier. On regarde donc si dedans il y a une occurrence de "image".

2.3.7 Étape 7 : exécution de sous-commande (–exec CMD)

La chaîne CMD passée en argument est parsée à la lecture des options du programme. Nous utilisons str_tok_r et str_tok avec "|" et " " comme séparateurs. Le résultat est stocké dans une structure contenant des argv pipés entre eux.
Pour l'exécution, nous insérons d'abord le nom du fichier dans la commande puis nous effectuons un fork par argv suivi d'un execvp, en redirigeant l'entrée et la sortie si besoin.

2.3.8 Extension 1 : Utilisation de libdl (-i)

Si l'option -i est passée en arguments, le chargement de libmagic se fait à la lecture des options. La mémoire est ensuite libérée une fois qu'on n'en a plus besoin, c'est à dire après la recherche récursive des fichiers.

2.3.9 Extension 2 : Utilisation de libpcre pour les expressions régulières PERL (-T)

Pour la recherche par expressions régulières PERL nous avons utilisé la librairie pcre2. Avec cette librairie il est possible de compiler le "pattern" (en PERL) , c'est-à-dire le motif qu'on veut retrouver dans un fichier (-T). Avec ce pattern compilé on va essayer de faire des "match" avec la chaîne de caractère d'intérêt. Si le résultat est supérieur ou égal à 1, une occurrence est trouvée et on peut afficher le bon fichier.

2.3.10 Extension 3 : Utilisation de fnmatch pour les patterns glob (–ename)

La difficulté ici, n'était pas d'implémenter la comparaison mais de trouver la bonne bibliothèque la faisant.

2.3.11 Extension 4 : Parallélisation des threads (-p N)

Nous avons étudié le fonctionnement des threads et implanté une parallélisation de **N** fonctions sur **M** threads "au brouillon", dans ./test/threadify.c. Cependant, nous avons préféré ne pas l'intégrer au programme principal car il ne nous restait pas de tests blancs supplémentaires et cela aurait pu introduire des sources d'erreurs.
De plus, la structure actuelle de notre programme ne nous permet pas de paralléliser l'analyse d'un fichier par les fonctions de libmagic. Elle nous permet seulement de paralléliser le traitement appliqué à ces fichiers.
Le gain potentiel étant trop faible face aux pertes envisageables, nous en sommes donc restés là.

2.3.12 Appendix 1 : Intégration Continue

Nous avons rapidement étudié le fonctionnement des tests d'intégration continue sur gitlab et nous avons créé notre *.gitlab-ci.yml*. Celui-ci ne fonctionnant pas et étant déjà arrivé en fin de projet, nous avons décidé de ne pas pousser son développement plus loin.

2.3.13 Appendix 2 : Documentation

Nous avons généré une documentation HTML simple à l'aide de Doxygen. Un raccourci(./doc.html.lnk) permet d'y accéder.

2.4 Difficultés rencontrées et résolution

2.4.1 Structure

Au début notre code était assez dense et peu organisé ce qui posait problème lorsque nous voulions implémenter les options. En effet quand nous avons voulu faire l'option –name où il ne faut afficher que les "bons" fichiers, tous les fichiers s'affichaient. C'est donc pourquoi nous avons aboutit à la structure que nous avons actuellement où le code est assez modulé pour faire en sorte que seuls les bouts de code qui ont directement lien soient placés dans les mêmes fichiers.

3 Tests

Nous avons créé un script bash(./auto_test.bash) en essayant de se rapprocher le plus des tests que vous faites. Pour cela, nous faisons un diff entre les deux commandes présentes dans le comments.txt (à l'exception de –ename où nous utilisons find -iname). Le résultat est ensuite stocké dans ./test_output(1 fichier par test). Par soucis de simplicité, on peut lancer les tests avec "Make test" qui va compiler et effectuer les tests.

4 Gestion de projet

Nous travaillions au début en faisant du pair programming afin de poser les fondations du projet. Ceci étant fait, nous nous sommes répartis les différentes fonctionnalités à implémenter.

4.1 Répartition du temps de travail

	Ishara	Samer
Conception	1h30	15h
Implémentation	30h	30h
Tests		2h
Rédaction du rapport	1h30	1h
Total	33h	48h

TABLE 1 – Tableau des heures de travail

5 Conclusion

Ce projet a été une opportunité excellente d'acquérir non seulement des connaissances sur POSIX et les systèmes UNIX mais aussi des compétences sur des outils tels que git, make, gitlab, doxygen. Il nous a également apporté de l'expérience sur la structuration d'un projet en C.