PROJET RS : RSFIND

Choix de conception

Nous avions démarré le projet avec l'idée de traiter les fichiers un à un. Cette solution nous a semblée peu adaptée au niveau de la complexité de l'algorithme, après relecture de certaines restrictions. Cela est d'autant plus vrai lorsque plusieurs conditions sont enchainées. Par exemple, une restriction –name suivit de -i sera beaucoup plus efficace si la recherche du type MIME des fichiers n'est appliquée que sur les fichiers ayant répondu aux caractéristiques de –name. Nous trouvions donc utile de rendre asynchrone le parcours récursif des dossiers et l'application des restrictions (également asynchrones entre elles). Nous avons donc choisi de minimiser les sollicitations du processeur (les calculs), au détriment de la mémoire RAM.

Ainsi, nous avons opté pour une structure récursive. Cette structure listOfFiles stock à chaque niveau une structure MyFile qui possède toutes les caractéristiques utiles pour faciliter le processus (Ex : Plus besoin de réutiliser des types DIR* et dirent pour savoir si un objet est un fichier ou non, la réponse à ce calcul est stockée au moment du parcours récursif). Nous avons également choisit cette forme en prévision de l'extension des multi-threads, elle nous aurait permis d'appliquer les restrictions sur la liste de fichiers de manière parallèle (Ex : Une fois que la restriction -n a fini d'être appliquée sur l'élément i, un système de sémaphore autorise le thread de la restriction -t à passer sur l'élément, s'il est encore existant dans la liste, sinon il passe à l'élément suivant).

Cette structure nous permet de gérer facilement les fichiers qui remplissent ou non les conditions requises par l'utilisateur de rsfind. Nous avons d'ailleurs ajouté un fichier texte *« structEXPLAIN.txt »* pour apporter plus de précisions sur l'utilisation de la structure.

Au niveau du projet, nous avons décidé de marquer trois parties, qui sont d'ailleurs très visibles dans le code de la fonction main :

- **argsHandler : Traitement des options et arguments passées à la commande rsfind.** On vérifie la validité des options, la cohérence de leurs arguments (Ex : éviter que ./rsfind –name -l soit exécuté avec -l retenu comme l'argument de name. Dans ce cas, le programme return 1 comme requis dans l'énoncé.). On vérifie également que le chemin sur lequel la commande rsfind est appliquée est celui par défaut ou bien un chemin fourni par l'utilisateur. On vérifie également que le chemin est valide.

- **searcher : Recherche récursive et application des restrictions à la structure.** On parcourt les fichiers et les dossiers de manière récursive, en remplissant notre structure récursive. Ensuite, on applique les restrictions en commençant par les plus restreignantes. Nous faisons ce choix pour optimiser le temps d'exécution en additionnant les restrictions.

- **optionToApply : Application des règles d'affichage de la sortie de rsfind.** On décide de la manière d'afficher. Cette partie gère également le multi affichage, en alternant les affichages pour chaque fichier, dans l'ordre : print, listing détaillé, résultat exec.

Organisation du travail

Nous avons décidé de définir 5 phases :

1. **Phase de compréhension** : Discussions ensemble du sujet pour comprendre les prérequis et tester des bouts de code sur les librairies proposées à l'étude dans l'énoncé du sujet rsfind.

2. **Phase de choix de structure** : Nous avions besoin de définir une base de code commune pour se répartir les options et ainsi faciliter la réconciliation de nos deux codes pour avoir un projet le plus cohérent possible. C'est ici que nous avons choisi l'implémentation en structure récursive.

3. **Phase de traitement des options :** Cette étape nous a pris un certain temps et peut être considérée comme une étape à part entière. Après quelques difficultés, nous avons décidé de séparer les problèmes : traiter la ligne pour repérer les mauvaises entrées par un processus fils, et si le fils ne repère aucun problème, le père effectue la lecture des options présente et réagit en fonction (variables booléennes, choix du chemin).

4. **Phase de restriction de la liste de sortie** : Grâce aux étapes précédente, nous avons pu traiter les options de manière indépendante. La structure récursive utilisée a été très utile pour implémenter de nouveaux besoins qui apparaissaient au fur et à mesure de l'implémentation des restrictions. Elle nous a également permis de traiter facilement les opérations de SUPPRESSION, AJOUT, DEPLACEMENT, qui auraient été plus compliquées avec une méthode de stockage moins optimisée.

5. **Phase de choix de la méthode d'affichage :** Après avoir défini la « liste restreinte » des fichiers remplissant les conditions de restriction, nous avons implémenté les méthodes d'affichage. Cette phase contient également – exec puisque cette option renvoie pour chaque fichier l'affichage d'une commande appliquée à ces fichiers.

À partir de cette structure de travail, le multi-option était plus facile à implémenter. De la phase 1 à 3, nous avons travaillé ensemble puisqu'il nous a semblé pertinent de définir une base commune avant de se répartir les tâches. Les phases 4 et 5 ont été développées en parallèle après avoir réparti les options à implémenter.

Difficultés rencontrées

Au cours du projet nous avons rencontré plusieurs difficultés à chaque phase. La première phase a été compliquée car il fallait réussir à comprendre le sujet dans sa globalité. Le plus dur a été de se lancer dans l'établissement d'un « plan d'action » pour gérer ce projet qui était de taille conséquente.

La 3ème phase était également longue car nous avions un problème, l'idée était de vérifier si la ligne d'option était scrupuleusement identique à un résultat attendu avant même de

récupérer les informations. Nous trouvions cela essentiel de séparer les deux actions pour éviter toutes erreurs, la solution mise en place a été de réaliser un fork pour éviter l'effet secondaire de la fonction getopt_long qui est de détruire le tableau d'arguments argv.

Au cours de la 4ème phase, l'option -t a pu être implémentée de manière presque exhaustive. Il y a notamment une difficulté de lecture des dossiers trop volumineux du fait que le buffer de lecture est un simple char qui insert chaque fichier en entier dans une simple chaine de caractère qui sera ensuite comparé avec la chaine entrée en paramètre. L'optimisation doit se faire par l'utilisation de threads qui traitent la lecture de chaque fichier indépendamment.

Au cours de la 5ème phase, nous avons eu des difficultés sur l'option de listing détaillé. Nous avions en effet des problèmes de version de langage (locale LANG). En effet, nous avons remarqué que le comportement à copier prenait en compte le locale LANG, il a donc fallu trouver un moyen de s'adapter au locale LANG utilisé par find (LC_TIME). Nous avons également eu du mal au niveau de l'option –exec, notamment au moment de diviser la chaine de caractère pour remplacer (ou non) les potentielles accolades présentent.

Temps de réalisation

Ceci est une estimation à postériori du temps alloué en fonction des phases décrites dans la partie « *Organisation du travail* ».

BENJAMIN SOULAN – [20 H]
JULIEN ELKAIM – [29 H]

Phase 1

Benjamin :
[6 H] – Codage d'entrainement sur les fonctions pour appliquer la théorie des manuels et les exemples d'utilisations plus ou moins optimisées présentes sur internet. C'est une sorte de pré mâchage intellectuel pour comprendre un maximum le fonctionnement des librairies et leurs utilisations possibles. Cela a facilité par la suite l'écriture du projet et évité des erreurs évidentes avec les connaissances acquises qui pouvaient alors être expliquées directement dans les commentaires des fichiers partagés.
Julien :
[3 H] – Lecture du sujet et discussions avec Benjamin pour établir un plan d'action.

Phase 2

Benjamin :
[4 H] – Début d'implémentation des premières fonctions de lecture récursives et de certains paramètres sans prendre en compte de la structure et des paramètres. Ces fonctions étaient donc le squelette des fonctions retrouvées dans le fichier optionToApply.c.
Julien :
[5 H] – Après avoir débuté la programmation sous un modèle plus simple (exécution fichier par fichier des différentes d'options de restriction et d'affichage), j'ai finalement rebroussé chemin et réfléchi à une structure pour optimiser la manière de traiter les données.

Phase 3

Benjamin :

[3 H] – Adaptation des fonctions précédemment créées à la structure prenant en compte les options.

Julien :

[2 H] – Comprendre la fonction getOpt et l'utiliser.

[2 H] – Vérifier la ligne d'option sans la détruire, choix d'utilisation de fork.

[1 H] – Implémenter la récupération des arguments, options, et installer la suite du code pour réagir aux options activées par l'utilisateur.

Phase 4

Benjamin :

[6 H] – Implémentation de l'option -l et -t, des difficultés rencontrées surtout sur la décomposition de la fonction ls -l -d, notamment sur l'affichage de la date qui demande une prise en compte des paramètres « locale » et d'une graphie différente entre l'anglais et le français (un espace est rajouté après le format du mois mais n'est pas implémenté automatiquement par locale…)

Julien :

[5 H] – Implémentation de l'option -i. Mauvaise relecture du sujet, recherche d'alternative à libmagic car n'étant pas présente nativement sur les machines.

[3 H] – Implémentation de l'option –name.

Phase 5

Benjamin :

[3 H] – Correction et aide de Julien sur les erreurs persistantes sur certaines fonctions, mplémentation du fichier test.sh.

Julien :

[2 H] – Implémentation de l'option --print (et de sa combinaison avec -l et –exec).

[6 H] – Implémentation de l'option –exec.