

Projet de RS 2018 : RSfind

Joly Clément, Fenouillet Lucas

21/12/2018

Introduction

L’objectif de ce projet était de réécrire nous-même une commande similaire à “find” par implémentation des éléments vu en cours. Les attendus du sujet et la nature du rendu nous ont poussé à aller plus loin dans l’écriture des fonctions de test de notre programme. Nous avons su implémenter la totalité de la partie obligatoire du sujet, et une majorité de la partie “bonus”. Ce document détaille notre programme et les choix d’implémentation que nous avons fait.

Structure globale

Notre code source se situe essentiellement dans le dossier “src” à la racine du dépôt. À l’intérieur nous avons toutes nos fonctions organisées de la façon suivante:

- Dans le fichier “main.c” se trouve la fonction `main()` de notre programme, ainsi que la gestion des options de la ligne de commande et la sélection des paramètres de lancement.
- Le fichier “list_dir.c” inclue la descente récursive de l’arborescence des dossiers, ainsi que l’appel à nos fonctions de filtrage et d’affichage.
- Chaque autre fichier source inclut l’implémentation d’une différente fonctionnalité du sujet, en tant que filtre ou affichage, et vient se greffer dans le reste du code par son appel dans les fichiers mentionnés plus haut.

Choix de conception et difficultés rencontrées

Nous expliquons ici les principaux choix et difficultés qui concerne la partie détaillée par étapes (partie obligatoire).

Tests

Les différentes étapes et fonctionnalités ont été testées localement par le script `test.sh`. Il exécute et compare un appel de notre fonction `rsFind` à un appel à

find pour différents arguments. Les commandes des tests sont stockées dans un format `.json` pratique à la racine du dépôt et les tests sont lancés en appelant le script `“test.sh”` dans un terminal. La création et le maintien de ces test aidaient à compléter les tests blancs fournis et étaient essentiels au développement du programme.

Étape 1: analyse des option de la ligne de commande

Comme conseillé dans le sujet nous utilisons la fonction `“getopt_long”`. Pour ce faire est utilisée une boucle sur les options de la ligne de commande, accompagnés de tests et d’une récupération d’argument lorsque leurs présence est requise. La difficulté de cette étape était principalement la familiarisation à l’appel d’une fonction qui ne nous était pas familière auparavant.

Étape 2-3: listing du contenu d’un répertoire et des sous-répertoires

La récupération du contenu d’un répertoire et le parcours de l’arborescence se fait par descente récursive, laquelle est écrite dans le fichier `“list_dir.c”` cette étape permet de construire un outil, que nous appelons le contexte, qui sert à donner le chemin d’exécution pour le reste des fonctionnalités (printers, filters) du programme. Cette étape a demandé une gestion pointilleuse de la mémoire et des appels récursif et avait pour principale difficulté la récupération du chemin correct vers les fichiers.

Étape 4: listing détaillé

Cette fonctionnalité est implémentée dans le fichier `“printers.c”`. La fonction qui l’implémente récupère le contexte de la recherche à l’aide des étapes 2 et 3. La difficultés majeure rencontrée à cette étape était d’avoir une sortie qui soit identique caractère-par-caractère à la sortie du `“ls -l”`. Non seulement le parseur et la récupération des informations devaient être pointilleux, mais un des champs de l’affichage détaillé (la date) dépendait de la langue utilisée par le système. Nous avons fait le choix de l’implémenter pour la version anglaise.

Étape 5: recherche de texte

Cette fonctionnalité est implémentée dans le fichier `“textSearch.c”`. Nous utilisons un parseur qui parcourt le fichier, ainsi qu’un système d’indices (annotés pointeurs dans le code) parcourants le mot à trouver dans le texte. Ces indices indiquent quelle part du mot à chercher dans le texte a été détectée à chaque instant, en gérant les éventuels sous-mots liés à une répétition de pattern.

Étape 6: recherche d’image

Nous détectons le type d’image avec la bibliothèque `libmagic`.

Étape 7: exécution de sous-commandes

Cette étape faisait partie des plus complexes des attendus obligatoires du sujet. Elle a demandé une grande part de temps pour son implémentation et les recherches qui lui sont liées. Cette fonctionnalité s'exécute en deux étapes: - La première le passage de la commande pour récupérer les arguments, insérer le chemin courant à la place des '{}', gérer les pipes - la seconde l'exécution de la boucle 'for' qui lance autant de 'fork()' que de sous programmes, dont les entrées/sorties sont interconnectées, nécessaires à la gestion des pipes.

Options

Thread

L'argument -p est interprété. Cependant, nous n'avons pas pu implémenter cette partie par manque de temps.

Chargement dynamique

Nous avons implémenté le chargement dynamique de `libmagic` avec `libdl`. Ceci a été testé avec l'intégration continue de Gitlab, en lançant le programme dans un environnement où la bibliothèque n'était pas installée.

-T & libpcrc

L'option -T a été implémentée. Cependant, elle ne fonctionne pas sur de gros corpus de fichiers à cause de corruptions de la mémoire.

Méta-caractères shell

L'option `--ename` de recherche avec les méta-caractères shell a également été codée.

Nombre d'heures passées sur les différentes étapes du sujet

Clément

Conception | 12h |
Implémentation | 41h |
Tests | 13h |
Rédaction du rapport | 2h |
Total: | 68h |

Lucas

Conception | 10 |

Implémentation | 40 |

Tests | 3 |

Rédaction du rapport | 4 |

Total: | 57h |