Projet Système
"find the cat"

Rapport de projet

Auteurs
Salim EL OUAFKI
Valentin CHANEL

Responsable de projet
Maiwenn RACOUCHOT

Octobre 2022-Décembre 2022

1 Choix de conception

1.1 Gestion des modes et programmation des commandes

Dans le but d'avoir une architecture itérative et facilement pouvoir rajouter des options, nous avons décider de créer une structure paramètre qui contient tous les flags et leurs paramètres.
On va donc populer la structure avec le parser avant de la passer à l'explorer, la partie qui effectue la recherche récursive.
Ensuite, nous utilisons un checker, qui prend en paramètre un fichier, la structure paramètre et qui renvoie un booléen selon que le fichier correspond au paramètres ou non.

1.2 Création du parser

Pour le parser, nous avons essayé différentes méthodes. Nous avons commencé par utiliser la fonction strtok pour stocker chaque partie de la commande (tokens) via la mise en place de délimiteurs mais celle-ci s'est avérée être assez limitée du fait que l'on considérait une chaîne de charactère sur laquelle nous effectuons nos opérations et non pas la ligne de commande directement.

C'est pour cela que nous nous sommes finalement rabbatu sur l'utilisation du paramètre argv[ ] et à un parcours qui s'opère via une boucle for avec un décalage suffisant (égal à 2 pour le paramètre et son argument) à chaque itération, mais cela posait problème pour les flags sans arguments.

Au final, nous avons choisi une boucle for incrémentée de 1 à chaque itération et avons mis en place une incrémentation interne à la gestion du paramètre quand celui-ci prenait un argument.

```c
#ifndef parameterStruct
#define parameterStruct
typedef enum { GREATER, SMALLER, EQUAL} modifier;

typedef struct {
    const char* name;
    long size;
    modifier sizeModifier;
    double timeSinceLastAcess;
    modifier timeModifier;
    const char* type;
    const char* regex;
    const char* patern;
    bool isDir;
    bool carrySearch;
} parameter;
#endif
```

Figure 1.1: Structure paramètre

```c
#ifndef util
#define util

// Coefficient de temps
#define min 60;
#define heure 60;
#define jour 24;

// Coefficient de taille
#define c 1;
#define k 1024;
#define M 1048576;
#define G 1073741824;
```

Figure 1.2: Stockage des coefficients

Nous analysons donc les arguments et peuplons la structure paramètre tout en faisaint un maximum d'error handling (figure 1.1). Nous utilisons également un fichier header util pour stocker les coefficients de temps et de taille afin de faciliter les conversions et de ne pas avoir à traiter les symboles multiplicateurs dans le parser(figure 1.2).

Pour chaque paramètre, nous avons implémentée des valeurs de type "boolean" afin de gérer le parcours de la chaîne de commande et d'assurer la terminaison du

programme si un paramètre ou un argument présentait un défaut. Le programme renvoie un message d'erreur explicant la source de celle-ci (voir figure 1.3).

```
    //Ici il faut vérifier que time est compris que de numéro hormis le premier si hasModifier et le dernier si not invalid
    for (int l = 0 + (int)hasModifier; l < strlen(time) - (int)(!invalid); l++)
    {
        if (!isdigit(time[l]))
        {
            param->timeSinceLastAcess = 0;
            param->carrySearch = false;
            printf("Erreur, la date ne doit être composé que de chiffre, hormis l'opérateur du début et un multiple à la fin\n");
            break;
        }
    }
    //printf("Date reconnu : %f\n",param->timeSinceLastAcess);
    i++;
}
else if(strcmp(argv[i],"-size")==0){
    if ( i + 1 >= argc)
    {
        param->carrySearch = false;
        printf("Erreur, pas d'argument après le paramètre -size\n");
        return param;
    }
```

Figure 1.3: Exemple de gestion des erreurs

1.3 Explorer l'arborescence

Pour analyser tous les fichiers, nous utilisons une fonction récursive qui, pour chaque dossier, itère sur tous les fichiers présent et les traitent. Si le fichier est un dossier, on appelle récursivement l'explorer dessus.

1.4 Vérifier les conditions

Tous les fichiers rencontrés vont être passer dans une fonction checker qui prend en paramètre divers informations du fichier ainsi que la structure paramètre. La fonction va succesivement faire les tests selon ce que contient paramètre et renvoyer vrai si le fichier rempli toutes les conditions et faux sinon.

Notre organisation est extrêmement pratique pour rajouter des conditions, on n'a même pas besoin de modifier le main ou l'explorer. Il suffit de parser la nouvelle condition, la mettre dans la structure puis de rajouter le test dans le checker.

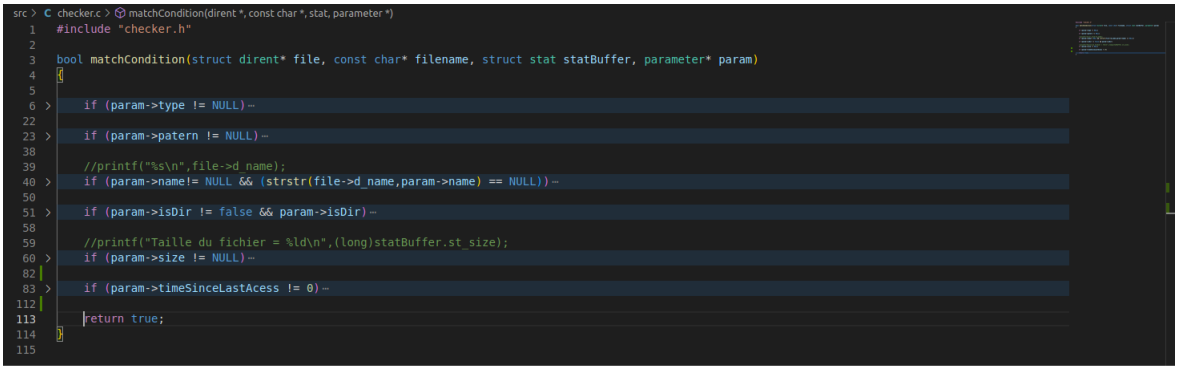

Figure 1.4: Structure de notre fonction de vérification

1.5 Difficultés rencontrées

Les principales difficultés auxquelles nous avons été confronté étaient la mise en place d'un parser sous la forme d'une boucle *for* qui parcours toute la chaîne de commande et qui comprend la gestion d'erreur ainsi que la correction de certains aspects du code qui engendraient une légère différence avec le résultat des fonctions test.

L'organisation de notre projet en différentes classes assez indépendantes a fait qu'il a été long de toutes les développer, de ce fait, nous avons du les assembler que tard dans le projet. Nous n'avons pas pu profiter des premiers tests automatiques. Malgré cela, cette stratégie n'a pas pour autant manquer de succès car une fois la structure mise en place, il a été très facile d'itérer dessus et de rajouter des fonctionnalités.

Nous avons également eu du des problèmes avec les tests automatiques qui ne fonctionnaient pas sur gitlab. Après étude, on s'est rendu compte que notre implémentation passaient les tests en local et que notre code était valide.

Ainsi, nous avons implémenter et passer les tests :

- test
- name, y compris avec les regex
- size
- date, sans les mots clef
- mime
- ctc
- dir
- parkour

1.6 Annexe

Voici des screenshots des tests passés localement ou sur gitlab où il y a eu des inconsistences :

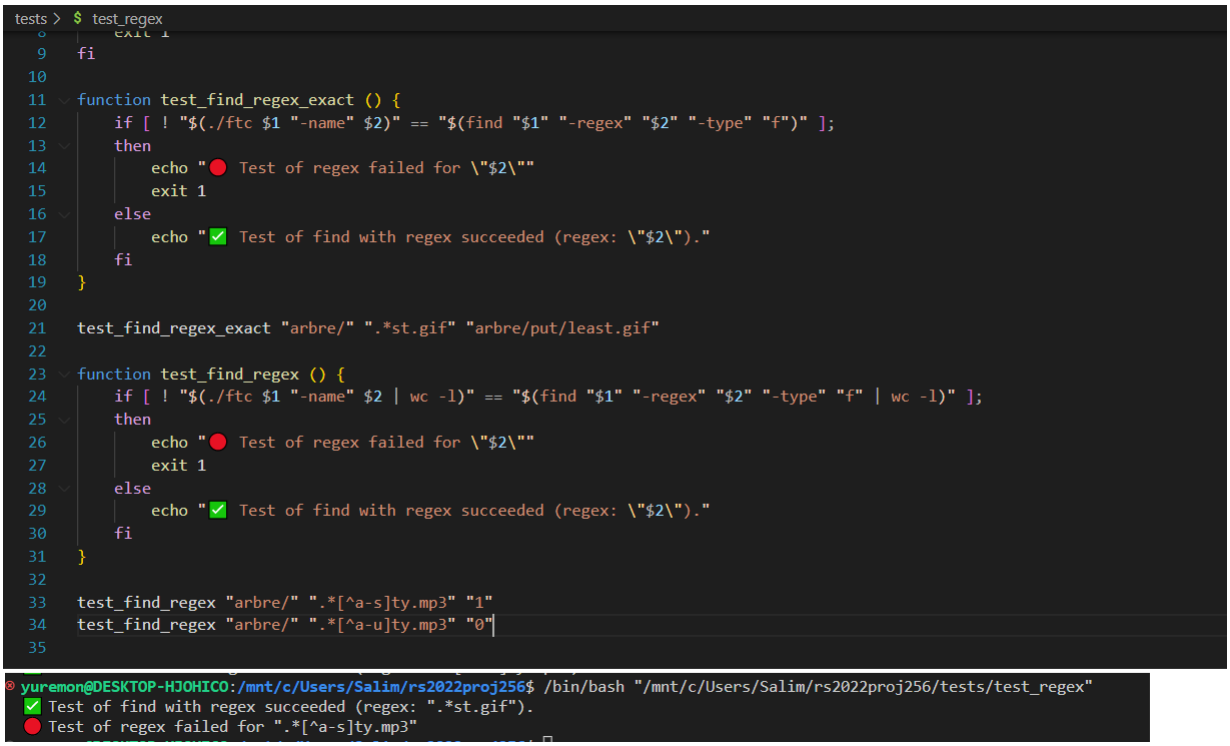

Figure 1.5: Résultat avec le test de base

Le test échoue quand aucun fichier ne correspond à la regex mais réussit quand au moins un fichier est valide.

Figure 1.6: Résultat avec un test différent (.gif à la place de .mp3)

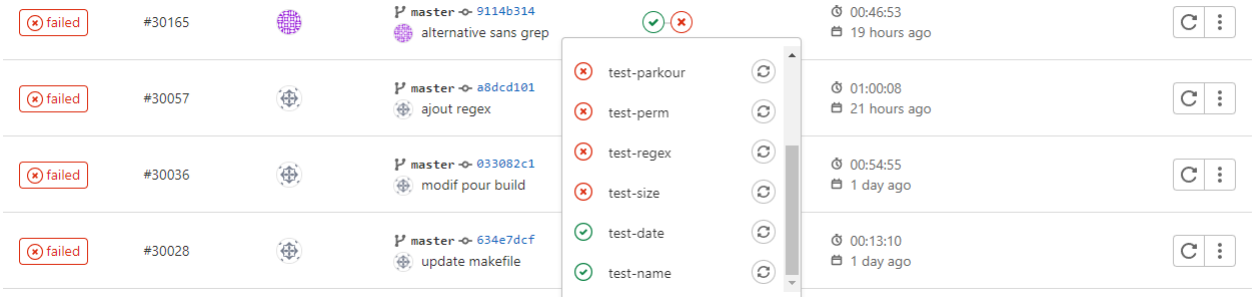

Figure 1.7: Test sur gitlab de date.

On voit sur cette capture d'écran effectuée le 18/12 que le test pour le paramètre -date est bien passée sachant que son code n'a pas été modifié depuis.