Project-1 OS

BENOUDA Haytam 572722

BADI BUDU Chris 569082

ANNAIM Marouane 572692

2023-2024

Contents

1 Introduction

Au sein de ce rapport, nous exposons en détail nos méthodes, incluant les processus, les pipelines, la gestion de la mémoire partagée, les signaux et l'exécution des exécutables. Nous décrivons également la mise en œuvre de notre code et abordons les problèmes que nous avons rencontrés au cours du projet. L'objectif principal de ce projet était de comparer une image donnée avec d'autres et de renvoyer celle qui présente la plus grande similarité.

2 Processus

Tout d'abord, nous avons commencé par créer 2 pipes qui permettront au père d'y écrire le chemin d'accès d'une image et aux deux fils de les lire : un pipe qui va du père vers le fils 1 et un deuxième du père vers le fils 2.
Pour pouvoir écrire dans chacun des pipes nous avons fait une boucle while pour lire depuis l'entrée standard (stdin) en utilisant la fonction fgets. Avant cela il faut initialiser un tableau afin de pouvoir stocker chaque chemin qui sont des chaines de caractères. Dans la boucle, on vérifie si la variable taille, qui est incrémentée à chaque itération de la boucle, est paire. Si c'est le cas, elle utilise la fonction write pour écrire la chaîne de caractères du chemin dans le descripteur d'écriture (WRITE qui permet d'écrire) du premier tube (pipe1) et vice-versa si c'est impair. Cela permettra d'avoir une exécution en concurrence.

3 Mémoire partagée

Nous avons initialisé la mémoire partagée de type struct pour pouvoir stocker le chemin de l'image et la distance. Nous actualisons la distance stockée dans la mémoire partagée si celle trouvée est plus petite que l'actuelle.
Cette approche optimise ainsi la complexité de notre code même si l'exécution prend énormément de temps à cause de img-dist. Comme vous pouvez voir dans l'Annexe1, la fonction mmap prend 6 arguments :

1. **NULL** : le système d'exploitation choisit une adresse appropriée pour vous.
2. `sizeof(struct image)` : La taille de la mémoire, en octets. Ici, n est le nombre d'éléments de type `struct image` que vous souhaitez allouer.
3. **protection** : C'est un entier qui autorise les lectures et l'écriture dans la mémoire.
4. **visibility** : C'est un entier qui spécifie les options de visibilité et de partage de la mémoire. Dans cet exemple, `MAP_SHARED | MAP_ANONYMOUS` est utilisé :
   (a) `MAP_SHARED` : Permet de partager cette portion de mémoire avec les enfants du processus père.

(b) `MAP_ANONYMOUS` : Fait en sorte à ce qu'on utilise pas les fichiers de cette mémoire.

5. **-1** : Spécifie que le fichier n'est pas utilisé.

6. **0** : C'est généralement utilisé pour spécifier un décalage dans le fichier à partir duquel mapper la mémoire. Dans ce cas, il est défini à zéro.

4 Comparaison

Dans cette fonction, nous créons un processus via un fork qui nous permettra de faire un appel système execlp pour exécuter le fichier exécutable img-dist dans le répertoire img-dist qui comparera 2 images. Execlp prend en paramètres le nom du fichier exécutable, les chemins d'accès des images à comparer et NULL pour spécifier que le dernier argument cité était la fin.
Lorsque cette exécution sera terminée, on sera de retour dans l'exécution du processus père qui récupèrera l'état de terminaison du fils grâce à un wait(status). Avec WEXITSTATUS(status), on connaitra la valeur de retour de img-dist qui est l'indice de différence entre les 2 images. On la comparera avec la valeur qui est stockée dans la mémoire partagée.

5 Bash

Par la suite, nous avons dû réaliser deux scripts en Bash : le *list-file* et le *launcher*. Le premier consiste à renvoyer la liste de tous les fichiers contenus dans un dossier, fourni comme seul argument dans ce script Bash. Il est également demandé de vérifier si le script reçoit bien uniquement le dossier en argument et que ce dossier existe bel et bien comme chemin d'accès. Pour lister seulement les fichiers contenus dans le dossier, nous avons utilisé une boucle *for* pour parcourir chaque élément contenu dans le dossier fourni. Nous vérifions ensuite si chaque élément est bien un fichier et non un sous-dossier. Ensuite, nous affichons cet élément en sortie, ce qui pourra être récupéré lors de l'appel de ce script.

Après cela, nous avons réalisé le script Bash *launcher*. Il a pour but de lancer *img-search* en proposant deux modes d'exécution différents. Nous avons commencé par modifier la variable `$PATH` pour y ajouter le chemin de *img-dist/* et nous avons donné les permissions d'exécution à ce chemin d'accès à l'aide de `chmod +x 'nom du fichier'`. Ensuite, nous vérifions si un troisième argument est fourni, représentant le chemin de la banque d'images *database_path*. S'il n'est pas fourni, sa valeur par défaut est le dossier *./img.*. Nous vérifions également si le nombre d'arguments fournis est correct (soit deux, soit trois arguments). Nous avons ajouté un mode d'aide avec la liste des commandes possibles pour utiliser le *launcher*.

Dans ce contexte, nous modifions le chemin de l'image à comparer passée en paramètre pour inclure son chemin d'accès complet. Si l'image donnée n'est pas un fichier existant, un message d'erreur est renvoyé.

Le premier mode d'exécution est le mode interactif. Il consiste à recevoir la banque d'images de manière interactive via le stdin. L'utilisateur devra donc entrer image par image celles qui constitueront la banque d'images et qui seront comparées à l'image demandée. Nous avons implémenté cela en vérifiant que le premier argument est bien `-i` ou `--interactive`, signifiant que l'utilisateur a choisi le mode interactif. Nous utilisons une boucle *while* qui lit tout ce que l'utilisateur entre dans la console via stdin et renvoie en sortie le chemin d'accès de l'élément qu'il a ajouté. La boucle *while* se termine à l'aide d'un *break* lorsqu'il appuie sur CTRL+D sans rien entrer comme image dans le stdin. Tous ces chemins que l'on récupère seront envoyés dans le stdin de *img-search* via un pipe. Ils seront récupérés par la suite dans le code source de *img-search* à l'aide de la fonction *fgets*. Le second mode d'exécution est le mode automatique (`-a` ou `--automatic`), qui reçoit directement toute la banque d'images. Comme mentionné précédemment, si elle n'est pas fournie, la variable database_path contient le chemin d'accès vers le dossier *img* par défaut. Nous utilisons le script *list-file* sur le dossier de la banque d'images pour renvoyer tous les fichiers qu'il contient au stdin de *img-search* via le pipe une fois de plus.

6 Signaux

Pour la gestion des signaux, nous avons implémenté une fonction handler qui va gérer nos deux cas de signaux. Si le signal est un signal SIGINT, on renvoie un message d'erreur signalant l'arrêt des processus et du programme, et l'on ferme tout à l'aide de l'exit(0). La même chose est faite pour le signal SIGPIPE qui, lui, renvoie un message d'erreur signalant la fermeture du pipe. L'appel de ces signaux est fait dans le code du processus père à l'aide de la fonction signal écrite de cette façon signal('nom du signal', handler) ; .

7 Problèmes

Comme dit dans l'introduction, nous avons rencontré quelques problèmes dans l'élaboration de notre et nous allons vous les présenter.
Comme premier problème nous avons eu des problèmes au moment où les processus fils souhaitaient lire dans le pipe pour recevoir le chemin d'accès de l'image à comparer avec l'image principale. Premièrement, nous n'utilisions pas de boucle while pour nous assurer d'avoir lu l'entièreté des chemins d'accès que le processus fils devait lire. Ensuite, la création des

processus fils se faisaient mal car le premier processus fils créé créait à son tour le 2e processus au lieu que ce ne soit que le père qui le fasse. Enfin, lors de la communication entre le père et le fils1 il nous fallait fermer le descripteur de lecture et d'écriture fils 2 afin d'éviter tout problème et vice-versa.
Durant notre travail, nous avons été confrontés à un problème de fichier, qui se voyaient être corrompus lors de leur utilisation. Un message d'erreur était renvoyé mais nous sommes parvenus à trouver comme solution d'utiliser un système de conversion des caractères à l'aide de dos2unix. Nous pensons que cela a été causé par le partage de notre travail via GitHub, mais sans pouvoir corriger cela autrement que l'utilisation de dos2unix. Nous espérons que le travail final remis ne soit pas confronté à ce problème sur la machine du correcteur.

8 Conclusion

En conclusion, nous avons élaboré le rapport d'OS détaillant les méthodes et les mises en œuvre utilisées tout au long de la réalisation de ce projet. Cela contribue à rendre notre programme plus compréhensible. Ce projet nous a offert une opportunité d'approfondir notre compréhension des concepts enseignés en cours.

9 Annexe

(a) Memoire partage

```
return mmap(NULL, size, protection, visibility, -1, 0);
```