RAPPORT DE LP25

SEMESTRE P21

BLANCHOT LUKAS,

CHAILLARD LÉO,

LIGNON THOMAS,

VIALA ALEXANDRE

Rapport LP25

I.Sommaire

II. Introduction

Ce projet a été réalisé durant le semestre de printemps dans le cadre de l'UV LP25 par Lukas BLANCHOT, Alexandre VIALA, Léo CHAILLARD et Thomas LIGNON. L'objectif de ce projet était d'écrire un programme qui va scanner récursivement les répertoires et enregistrer son l'arborescence dans un fichier. Pour cela nous disposions de certains éléments du programme, tels que des fonctions à utiliser et les structures pour enregistrer les informations des fichiers scannés.

III. Présentation des fichiers

Pour notre projet nous avions pour contrainte d'utiliser différents fichiers afin d'avoir une organisation claire. Nous avons choisi par la suite de ranger les fichiers .c dans un répertoire "c_files" et les fichiers .h dans un répertoire "include". De plus, notre makefile génère deux répertoires pour stocker les fichiers binaires et les fichiers objets, le répertoire "bin" et le répertoire "object" respectivement.

1. Makefile

Le makefile a été écrit de manière à ce que les informations des fichiers puissent être récupérées dans différents répertoires. Nous avons également fait en sorte que le makefile soit capable de créer des fichiers afin de ne pas avoir besoin de créer des répertoires vides ne servant à rien avant la compilation.

Nous avons également implémenté un nettoyage récursif des répertoires à l'aide de la variable $(RM) ainsi que divers phony targets pour faciliter le test du programme. Ces phony targets sont davantage détaillées dans la documentation doxygen et le readme.md.

Enfin nous avons eu recours à des règles génériques pour générer les fichiers objet. Cela a permis de raccourcir largement la taille de notre makefile.

2. Fichier main

Il s'agit du fichier principal de notre programme. Nous l'utilisons uniquement pour détecter les options passées en arguments du programmes à l'aide de la fonction getopt et pour faire tourner l'ensemble des fonctions du programme. Nous avons préféré utiliser un `getopt_long` plutôt qu'un getopt classique afin de faciliter la compréhension des options par l'utilisateur. En prenant des options longues, nous prenons le risque que l'utilisateur essaie de faire tourner le programme avec une syntaxe non désirée :

Ici, l'utilisateur veut faire tourner le programme sur le répertoire Images du répertoire personnel de l'utilisateur. Cependant ~ n'est pas interprété dans ce cas car nous utilisons des guillemets. Pour être sûr que l'on ait un fonctionnement normal, il faudrait utiliser des anti-quotes.

Une fois que les options seront récupérées, le programme pourra faire tourner les fonctions principales du programme. C'est-à-dire la sauvegarde dans la mémoire vive du PC de la structure de fichiers puis la sauvegarde dans un fichier choisi par l'utilisateur.
Nous avons enfin ajouté un message d'aide pour permettre à l'utilisateur de mieux comprendre l'utilisation du logiciel. Pour cela, nous nous sommes inspirés du manuel pour rédiger le comportement du programme en fonction des options passées.

3. Fichier scan

Le fichier scan est composé des fonctions permettant de faire un scan récursif de l'arborescence de fichiers. Le but dans ce fichier est de remplir une structure `s_directory` avec toutes les informations des fichiers. Cette structure est chaînée triplement avec un pointeur sur le répertoire suivant, un pointeur sur un sous-dossier et un pointeur sur une structure `s_file` qui permet de stocker les informations d'un fichier.

La fonction `process_file()` permet de scanner un fichier et de stocker le résultat de ce scan dans une structure `s_file`. Cet élément est directement renvoyé sous la forme d'un pointeur. Cette fonction utilise la fonction `stat()` afin d'obtenir les informations sur le fichier.

La fonction la plus importante du fichier est évidemment la fonction `process_dir()` qui va traverser un répertoire en utilisant la fonction `process_file()` et va ensuite faire un appel récursif à elle-même sur tous les répertoires à l'intérieur du répertoire courant afin de traverser entièrement l'arborescence de fichier.

Le fichier est également doté de deux autres fonctions : `catPath()` et `getRelativePath()`. Ces fonctions sont là pour permettre de construire le chemin complet d'un fichier (`catPath()`) ou d'obtenir le nom de base du fichier (`getRelativePath()`).

4. Fichier save

Le fichier save est composé de diverses fonctions permettant de sauvegarder le contenu d'une structure `s_directory`, représentant un répertoire, dans un fichier précisé. De base ce fichier a pour nom `"yyyy-MM-dd-hh:mm:ss.scan"` et est stocké dans un répertoire caché appelé `".filescanner"`.

Les fonctions utilisées sont pour la plupart récursives et permettent de traverser la structure avec des algorithmes relativement simples. La première fonction `save_to_file()` est une fonction qui va simplement appeler les autres fonctions du fichier après avoir créé tout ce qui était nécessaire (création du répertoire `.filescanner` s'il n'existe pas, du fichier scan, etc.).

Le programme repose ensuite sur la fonction `write_files()` et `write_directories()`. La première fonction est utilisée pour afficher tous les fichiers (qui ne sont pas des répertoires) dans le fichier et la deuxième fonction est utilisée pour afficher les répertoires et leur contenu de manière récursive.

Ces deux fonctions utilisent des fonctions plus courtes afin de fonctionner correctement. La première fonction est `print_tabs()` qui permet d'écrire un certain nombre de tabulations dans un fichier, ce qui rend la lecture des répertoire plus lisible. A

cela s'ajoute une seconde fonction permettant d'afficher la somme md5. En effet la somme md5 est représentée sous la forme d'un tableau de `unsigned char`. Or il n'existe en langage c aucun moyen d'afficher un tableau de `unsigned char` comme c'est le cas pour un string par exemple. On doit donc passer par une boucle for qui va traverser la somme md5 afin d'afficher son contenu octet par octet.

5. Fichier tree

Le fichier tree est composé de fonctions permettant d'ajouter des éléments aux structures chaînées de notre programme, ou bien d'en nettoyer le contenu.

Tout d'abord, les fonctions `append_subdir()` et `append_file()` qui s'occupent respectivement d'ajouter un élément en tête des listes de type `s_directory` et `s_file`. Puis la fonction `clear_files()` qui s'occupe de supprimer les éléments de la liste de type s_file contenue dans notre structure de données. Et enfin, la fonction `clear_subdirs()` qui, à l'aide de `clear_files()`, nettoie récursivement le contenu des sous-répertoires du répertoire courant.

Nous avons notamment eu recours à l'utilisation de ces fonctions dans le fichier scan afin de construire nos structures de données, puis à la fin du fichier main pour libérer leur contenu.

6. Fichier md5sum

Le fichier md5sum est composé d'une unique fonction qui permet de calculer la somme md5 d'un fichier. Cette fonction utilise les fonctions de la bibliothèque md5.h d'openssl. Nous utilisons notamment `MD5_Init()`, `MD5_Update()` et `MD5_Final()`. Ces fonctions ont la particularité de modifier les paramètres qu'on leur donne lorsqu'un fichier est ouvert dans la pile du programme.

Ce fichier a posé quelques problèmes car la somme md5 ne peut pas être calculée si certaines conditions ne sont pas remplies :

- Le fichier ne peut pas être une fifo. En effet, lors de l'un de nos tp nous avons utilisé des fifo pour communiquer aux programmes des informations. En tentant de calculer la somme d'une fifo, nous nous sommes rendu compte que le programme tournait indéfiniment sans s'arrêter. Il se trouve qu'un fichier fifo est l'équivalent d'un tube et il est donc impossible d'en lire le contenu. Nous avons donc fait en sorte que notre programme ne calcule pas la somme md5 si il scanne un fichier fifo.
- Le fichier ne peut pas avoir des droits d'accès restreignant la lecture. En effet, si le fichier que l'on tente de scanner n'a pas des droits de lecture alors il est impossible de l'ouvrir. Or le calcul de la somme md5 se base sur l'ouverture d'un fichier en mode lecture pour fonctionner. Nous ne calculons donc pas la somme md5 dans un tel cas.

IV. Méthode de travail

Nous faisions des réunions toutes les 2 semaines comme cela nous avait été demandé dans l'énoncé du projet. Nous ne faisions pas toujours des réunions avec tous les membres du groupe afin d'optimiser notre temps.

1. *Réunion 1*

Lors de la première réunion nous avions décidé pour commencer que :

- Alexandre et Leo feraient les fonctions du fichier tree.
- Thomas ferait le scan.
- Lukas s'occuperait du save.

Pendant cette réunion nous avons revu le sujet et programmé la prochaine réunion.

2. *Réunion 2*

Lors de la deuxième réunion nous avons décidé qu'il serait plus productif pour notre groupe de faire des groupes de 2 afin que ceux ayant le plus de difficultés puissent quand même avoir un impact sur le projet.
Nous avons ensuite fixé les objectifs pour la réunion suivante :

- Alexandre et Lukas avanceraient sur le fichier save
- Léo et Thomas géreront le fichier scan.

3. *Réunion 3*

Pour la troisième réunion nous avons décidé qu'il faudrait commencer la rédaction du rapport, améliorer les éléments déjà ajoutés et commencer le md5.
La répartition était la suivante:

- Lukas et Léo commenceraient le rapport
- Thomas et Alexandre feraient le md5.

Et en parallèle, toujours en groupe de 2, on améliorerait le code déjà existant.

4. *Langage utilisé*

Pour réaliser l'ensemble de nos fichiers nous avons décidé d'utiliser principalement l'anglais pour nos noms de variables et pour l'ensemble de de notre documentation. Il s'agit du langage universel et il est donc plus adapté pour réaliser des programmes modulables et modifiables au cours du temps.

5. *Documentation*

La documentation a été réalisée à l'aide du logiciel Doxygen et de son interface Doxywizard. De plus nous avons utilisé l'extension "`bbenoist.doxygen`" de Visual Studio Code qui nous a permis de créer facilement les commentaires doxygen.

Nous avons également modifié le README.md fourni avec l'énoncé du sujet et l'avons ajouté sur la page d'accueil de notre documentation Doxygen.

V. RETEX

1. *Expérience de Thomas*

Après un semestre complet sans avoir fait ce C, les TD et TP se sont montrés plus qu'utiles pour m'aider à réaliser ce projet. Mais même avec cela j'ai souvent eu recours à l'aide de mes camarades pour que tout fonctionne bien comme je le souhaitais.

Le fait de travailler en groupe de 2, nous a permis de bien avancer sans qu'aucune partie ne prenne trop de retard sur les autres.

Certes tous les éléments bonus n'ont pas été ajoutés mais je pense que ne pas tenter de les ajouter est préférable pour que nous puissions nous préparer en avance à la préparation du rapport et de la présentation orale.

2. *Expérience de Lukas*

Le projet fut très difficile mais aussi très bénéfique pour moi. Je n'étais pas très à l'aise avec le C et cela m'a permis de dépasser certaines limites. De plus les TD et TP se sont montrés très utiles pour me mettre au niveau.

Nous avions décidé de répartir un travail chacun selon les capacités de tout ce qui a permis une régularité constante très bénéfique.

Ce projet a consisté en une approche concrète du métier d'ingénieur et fut pour moi très enrichissant. En effet, la prise d'initiative, le respect des délais et le travail en équipe seront des aspects essentiels de notre futur métier.

Les principaux problèmes, que j'ai rencontrés, concernaient mon manque de connaissance du langage car ce fût pour moi le semestre ou je l'apprenais.

3. *Expérience de Léo*

Projet qui a été pour ma part aussi difficile que bénéfique. En effet, difficile de part la rigueur nécessaire pour manipuler la mémoire à l'aide du langage C; bénéfique car il a été un très bon moyen de solidifier les connaissances acquises lors de l'UV LP25.

L'avancement sur le fichier scan m'a personnellement opposé une certaine difficulté. Bien que la logique employée ait été relativement facile, la précipitation a eu pour cause de me rappeler la consciencieusité nécessaire pour coder en langage C.

Je suis d'avis que nous avons eu une bonne organisation en tant que groupe. Nous nous sommes rapidement mis au travail, et avons correctement réparti le travail selon la volonté et les capacités de chacun, ce qui nous a permis d'avancer à une vitesse régulière sur ce projet.

4. *Expérience de Alexandre*

Ce projet était de moyenne envergure mais il y avait tout de même beaucoup de concepts complexes que nous avons abordés en cours et qui nous ont resservi au cours du projet. Cela nous a permis d'appliquer ce que nous avions appris mais nous nous sommes

tout de même heurtés à des difficultés. J'ai eu du mal notamment à comprendre le fonctionnement des sommes md5 dès le début. Il a donc fallu que j'apprenne correctement à me servir de gdb afin de me faciliter grandement la vie. Une fois cela fait, le projet s'est déroulé bien plus facilement.

Nous avions une bonne entraide dans le groupe et nous nous retrouvions souvent à déboguer ensemble. Cela nous a permis de connaître l'ensemble du programme et d'avancer plus vite en partageant nos connaissances sur le C.

A l'issue de ce projet, certaines fonctionnalités ne sont pas parfaites. Nous aurions pu par exemple créer une page de manuel pour notre programme au lieu d'utiliser une option -h. Nous aurions également pu inclure les bases de données sqlite mais par faute de temps, nous ne l'avons pas fait.

Dans l'ensemble je suis plutôt satisfait du travail que nous avons accompli.