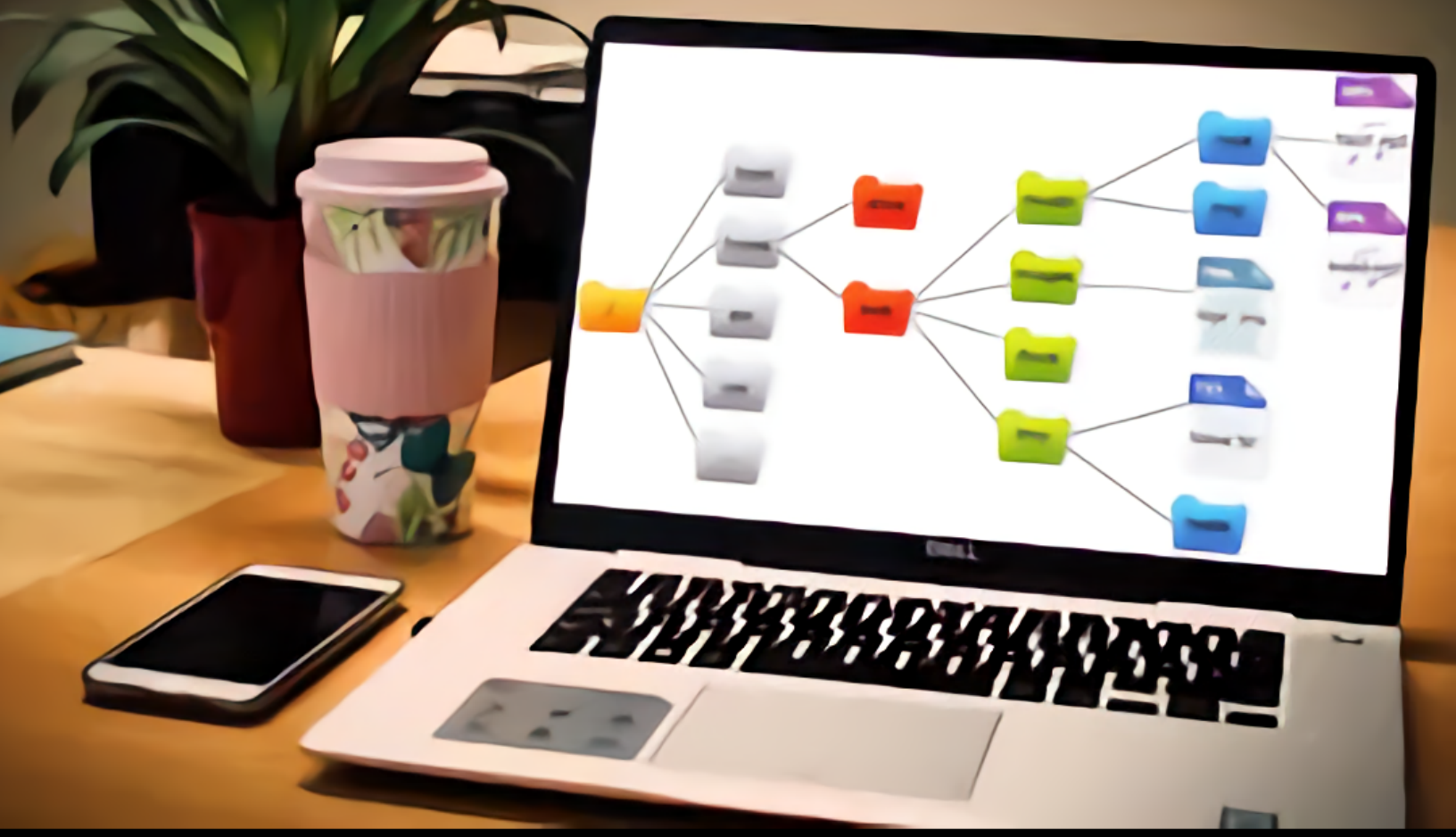

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Rapport projet LP25

Printemp 2021

responsable d'UV : Oumaya BAALA CANALDA

Adrien Burgun
Eléanore Renaud
Oscar Dewasmes
Jean-Maël Legrand

UTBM

I. Introduction

Ce projet avait pour but de créer un programme qui parcourt de manière récursive une arborescence de fichier. Durant ce parcours, il va stocker les informations sur tous les fichiers dans une liste chaînée. Une fois que le programme aura parcouru tout le fichier il va transcrire la liste chaînée dans un fichier nommé en fonction de la date et de l'heure.

II. Le projet

1. Contenu du projet

Le projet à la structure suivante :

- Un dossier "include" avec les fichiers header.
- Un dossier "src" qui contient les fichiers sources.
- Un dossier "test" qui contient les tests.
- L'application compilé et les fichiers objets se trouveront dans un dossier "build" par défaut, l'exécutable sera ./build/bin/projet
- Un Makefile
- Un CMakeLists.txt
- Un readme avec les instructions nécessaires pour l'exécution du programme et du projet

2. Structures et types utilisés dans le projet

Pour réaliser ce projet, il était nécessaire d'utiliser des types prédéfinis et certaines structures imposées. Il y avait donc deux structures et une énumération imposées pour contenir l'arborescence. Ainsi, une structure était adaptée aux répertoires et une aux fichiers. Ces structures sont stockées dans le fichier defs.h.

```
typedef enum {DIRECTORY, REGULAR_FILE, OTHER_TYPE} e_type;

typedef struct _file {
    e_type file_type;
    char name[NAME_MAX+1];
    time_t mod_time;
    uint64_t file_size;
    u_char md5sum[MD5_DIGEST_LENGTH];
    struct _file *next_file;
} s_file;

typedef struct _directory {
    char name[NAME_MAX+1];
    time_t mod_time;
    struct _directory *subdirs;
    s_file *files;
    struct _directory *next_dir;
} s_directory;
```

L'énumération *e_type* permet d'expliciter les trois types d'éléments traités et de s'y référer par leur nom et non par leur valeur numérique.

Pour ce qui est des structures, leur définition est similaire à l'énumération. En effet, les fichiers nécessitent plus de variables pour stocker des informations et les répertoires doivent contenir leur arborescence.

3. Les fichiers principaux

Le programme est séparé en plusieurs fichiers de code, chacun ayant un rôle précis.

main.c

Ce fichier contient la fonction main qui lit les options passées au programme, en utilisant getopt. Elle permet également d'appeler les fonctions des autres fichiers pour scanner le répertoire cible, construire la structure en mémoire, ainsi que l'écrire dans le fichier scan et libérer la mémoire allouée.

scan.c

Ce fichier, associé à scan.h, contient les définitions des fonctions nécessaires au scan, notamment :

```
s_directory *process_dir(char *path, bool md5sum, bool verbose);
s_file *process_file(char *path, bool md5sum);
```

Ces deux fonctions lisent respectivement les dossiers et les fichiers; elles prennent toutes les deux en entrée le chemin vers le dossier/fichier (*char* path*), allouent les structures *directory_t* et *file_t*, les remplissent et les retournent.
La fonction *process_dir* est récursive et s'appelle dès qu'elle rencontre un sous-dossier.

Par rapport au prototypes des fonction du fichier scan.c proposés par le sujet, nous avons ajouté deux booléens :

- *md5sum*, qui indique si la fonction doit calculer la somme md5 des fichiers rencontrés.
- *verbose*, qui indique si la fonction doit décrire son déroulement dans la console.

De plus, une fonction qui retourne le dernier élément d'un chemin de fichier a été écrite pour alléger les deux fonctions principales du fichier scan.c:

```
// Splits the input with `separator` and returns a pointer to the last element
const char* get_basename(const char* input, char separator);
```

save.c

Le fichier save.c contient les fonctions permettant de sauvegarder le contenu des structures *directory_t* et *file_t* dans un fichier. La fonction principale de ce fichier est *save_to_file*, qui ouvre le fichier de sortie (pointé par *path_to_target*) et appelle la fonction récursive *save_to_file_recursive*.

Cette dernière parcourt de manière récursive la structure *directory_t* et écrit son contenu ligne par ligne dans le fichier de sortie, sous un format imposé:

- *0* <*date*> <*chemin*>*/* : pour un répertoire
- *1* <*date*> <*taille*> <*MD5*> <*chemin*> : pour un fichier standard; la valeur MD5 est remplie par des zéros si MD5 n'est pas activé et la taille est en octets
- *2* <*date*> <*chemin*> : pour un autre type de fichier (lien symbolique, etc.)

Trois fonctions auxiliaires se chargent de formater chaque type de ligne:

- `bool construct_dir_line(FILE* output, s_directory dir, const char* path_to_parent_dir)` formate une ligne pour un dossier;
- `bool construct_file_line(FILE* output, s_file file, const char* path_to_parent_dir)` formate une ligne pour un fichier normal;
- `bool construct_other_line(FILE* output, s_file file, const char* path_to_parent_dir)` formate une ligne pour un autre fichier;

tree.c

Le fichier tree.c contient les fonctions permettant de manipuler les structures `s_directory` et `s_file`.

```
bool append_subdir(s_directory *child, s_directory *parent);
```

Cette fonction ajoute la structure *child* à la liste chaînée de dossiers dans *parent->subdirs*. La fonction retourne *false* en cas de problème et *true* sinon.

```
bool append_file(s_file *child, s_directory *parent);
```

Cette fonction ajoute la structure *child* à la liste chaînée de fichiers dans *parent->files*. La fonction retourne *false* en cas de problème et *true* sinon.

```
void clear_files(s_directory *parent);
```

Cette fonction libère la mémoire des fichiers contenus dans le répertoire parents. *parent->files* prend la valeur *NULL* à la fin de l'exécution de cette fonction.

```
void clear_subdirs(s_directory *parent);
```

Cette fonction libère uniquement les répertoires et fichiers dans *parent->subdirs*, qui prend ensuite la valeur *NULL*. Elle n'est pas utilisée, au profit de la fonction *free_dir*, qui libère l'entièreté de la structure.

```
void free_dir(directory_t* dir)
```

Cette fonction libère récursivement *dir*, ses sous-dossiers et ses fichiers.

md5sum.c

Ce fichier contient la fonction *compute_md5* et le code nécessaire pour fonctionner avec soit la librairie *openssl/md5.h*, soit la librairie *bsd/md5.h*.

```
compute_md5(char *path, uint8_t buffer[])
```

Cette fonction calcule la somme MD5 d'un fichier donné en paramètre (`char *path`) et stocke ainsi cette valeur dans un tableau de caractère (`uint8_t buffer[]`). Elle renvoie *false* en cas d'erreur et *true* en cas de succès.

Le makefile

Le makefile permet la compilation du projet dans un dossier “build”, avec l’exécutable dans le répertoire “build/bin” et les objets dans “build/objects”.

Les fichiers sources sont automatiquement récupérés avec une wildcard dans le dossier “src” et les headers sont récupérés dans le dossier “include”.

Il est possible d’indiquer les fichier sources à exclure avec la variable “SOURCES_IGNORE“, le nom de l’exécutable avec la variable “EXECUTABLE“ (par défaut: projet), le dossier source avec “SOURCE_DIRECTORY” (par défaut: src) ainsi que le dossier des header avec “HEADER_DIRECTORY” (par défaut: include).

Le build directory contenant les dossiers “bin” et "objets" peut être spécifié avec la variable “BUILD_DIRECTORY” (par défaut: build).

Après avoir remarqué que OpenSSL n’était pas installé par défaut sur les machines utilisées pour tester, nous avons décidé de permettre au code d’utiliser soit les fonctions MD5 d’OpenSSL, soit les fonctions MD5 de BSD. Pour pouvoir dire au code quelle librairie utiliser, il est nécessaire d’indiquer la librairie utilisée lors de la compilation avec "OPENSSL_MD5=1” ou "BSD_MD5=1“, si les deux sont indiqué alors openssl prévaut.

Les targets sont :

- all/build : Le target par défaut du makefile est build qui permet la compilation du projet.
- clean : supprime les fichiers objets
- clean-all : supprime les fichiers objets, l’exécutable et le dossier build en optionnellement.
- run : lance le programme avec les arguments de la variable “APP_ARG”
- run-verbose: même chose que run mais lance le programme en verbose par défaut
- help: affiche l’aide du makefile

4. Les tests automatisés

Nous avons décidé de tester le code de manière automatisée, afin de réduire le risque de bugs et de pouvoir s'assurer que des changements du code ne le cassent pas. Pour se faire, nous avons utilisé la librairie check, qui est une librairie contenant des outils standards pour tester du code: assertions, suite de tests, messages d'erreur compréhensifs et gestion des crash. La suite de test se trouve dans le dossier *test/* et peut être compilée avec *cmake*; les instructions sur la compilation se trouvent dans le README.

Les différentes fonctions du code source principal ont été testées séparément, ce qui nous a permis de nous assurer que chaque composant du code fonctionnait correctement avant de les combiner ensemble.

Ces tests nous ont permis de déceler plusieurs bugs qui n'auraient autrement pas ou difficilement été trouvés:

- La fonction *get_basename* retournait *NULL* lorsque la chaîne en entrée ne contenait pas de slash, ce qui causait par la suite un segfault (corrigé avec le commit 859c8c5).
- La fonction *process_directory* ajoutait les dossiers à la liste des fichiers (découvert avec le commit 5f383e3 et corrigé avec le commit 9b860d7).
- La fonction *save_to_file* affichait le nom du dossier en double (corrigé avec le commit 85c0e81).

III. Le RETEX

Difficultés rencontrées

Le linkage avec openssl n'était pas facile et les noms des headers ne correspondaient pas aux noms des shared objects. Écrire les tests avec check prenait beaucoup de temps également.

Pour pouvoir transformer la date de modification du fichier en String, dans la partie de sauvegarde, il a fallu lire énormément de document sur ce sujet pour comprendre qu'il fallait transformer la date depuis le type time_t en tm avec la fonction localtime() afin de pouvoir utiliser la fonction strftime().

De plus, lorsqu'il a fallu tester la fonction de sauvegarde, nous avons dû écrire une structure de données à la main sans l'initialiser à zéro. Ce qui a par la suite posé des problèmes de fautes de segmentation.

Ce problème a finalement été réglé, au moment où nous avons écrit une meilleure procédure de test. Il n'y avait alors plus d'erreurs.

Retour d'expérience

Eléanore

Ce projet a été une expérience très enrichissante du fait de pouvoir travailler dans un groupe de quatre personnes. Le sujet me paraissait plutôt abstrait au tout début, peut-être parce que mon premier projet en C était le jeu de la Belote et que l'idée de concevoir un jeu auquel je pouvais jouer m'intéressait beaucoup. Si je devais changer quelque chose, je pense que ce serait la gestion du temps pour ce projet qui a été assez difficile à gérer parfois.

Adrien

J'ai pu découvrir avec ce projet la librairie *check* et ait eu l'occasion de déceler et corriger des bugs qui n'étaient pas triviaux. J'ai plus apprécié ce projet par rapport à celui de la Belote (en LO21), car il a nécessité plus de mes compétences et utilisait des notions plus avancées de C.

Oscar

Depuis ma première découverte du C, j'ai beaucoup apprécié le concept d'algorithme récursif. C'est pourquoi j'ai beaucoup aimé, et que j'ai trouvé très naturel la conception des fonctions de scan et de sauvegarde qui sont récursives.

Jean-Maël

Ce projet a été très intéressant pour moi notamment la découverte des librairies qui me sont nouvelles et j'ai également pu étendre mes connaissances sur les makefiles.
J'ai également apprécié le fait que ce projet utilise des notions plus avancées de C.

Bilan

Au début du projet, la division des tâches et actions s'est faite assez rapidement, ce qui nous a garanti un travail efficace qui nous a permis de répondre aux attentes du projet et de finir dans le temps imparti. Cette avance de temps nous a permis de résoudre les derniers bug d'erreur et de bien tester le programme.

Comme quasiment toutes les personnes du groupe savaient déjà utiliser Git, l'organisation a été d'autant plus efficace. Pour l'organisation du travail, nous avons réalisé des réunions par visioconférence et non en présentiel, ce qui n'a pas posé de problème pour la réalisation du projet.

Pour ce qui est de la réalisation même du projet, la résolution des bugs n'a pas été facile car la plupart d'entre nous étaient habitués à utiliser les débogueurs intégrés aux IDE comme Visual Studio. Mais nous avons su nous adapter et passer outre ce petit détail.