UE Prog5 – Projet Logiciel – Compte Rendu

Groupe 16

CHALAS / DAVID / FAUCHE / JOONEKIN- -ARNAUD / MARINHO / SANCHEZ

Table des matières

I – INTRODUCTION

Premièrement nous allons voir la forme de nos structure avant de commencer à rentrer dans les fonctions et leurs fonctionnements.

1 – Nos structures

Pour réaliser ce projet nous avons utiliser les structures fourni par le fichier elf.h et avons lu les octets par la méthodes fread.

Notre structure principal se compose de plusieurs autres structures. Notre première problématique était l'allocation en mémoire. Nous souhaitions une allocation dynamique. Nous avons donc opter pour une construction par liste chaînées pour stocker les élements. Voici notre principale structure :

```
Ofile {
        Elf_Header
        SecHead
        SymTab
        StringTab
        StringTab
        ListReimpTab
        FILE
}
```

Cette dernière est composé de plusieurs autres structures comme dit précédemment. Ces dernières se ressemble énormément.

On retrouve deux type de structure, la première qui consiste uniquement à sauvegarder une nombre d'élément fixe :

```
Elf_Header {
        //Structure de elf.h utilisé pour le header 64bits et que
        l'on utilise également pour le 32bits
        Elf64_Ehdr headformat

        //Entier nous donnant l'information de l'architecture
        (32bits, 64bits). Ce dernier est présent dans headformat
        mais nous préférons créer un entier pour faciliter l'accès
        de ce dernier.
        int architecture

        //Entier nous donnant l'information du format d'écriture
        du fichier (bigedian / littleendian)
        int endianess
}
```

La seconde quand à elle consiste à récupérer un nombre indéfinis d'élément. Ces éléments en question on également un nombre fixe de sous-éléments comme dans la structure précédente :

```
typedef struct _OneHeader OneHeader;
struct _OneHeader
        {
        //Structure de elf.h utilisé pour la table de section
        header 64bits et que l'on utilise également pour
        le 32bits
        Elf64_Shdr tableformat;
        //Suivant de la liste chaînée
        OneHeader * suivant;
};
typedef struct
{
        //Tête de la liste chaînée
        OneHeader * tete;
        //Nombre d'élément dans notre liste chaînée
        int nb;
} SecHead;
```

Cette structure est également utilisé pour la table des symboles grâce à la structure **Elf64_Sym** de elf.h mais également pour une table de réimplantation avec la structure **Elf64_Rel**.

Mais aussi pour les tables des strings avec un simple char en liste chaînée. Nous avons deux tables des strings donc deux structures **StringTab**.

Mais nous pouvons avoir plusieurs table de réimplantation, pour cela nous utilisons le même genre de structure en faisant une liste chaînée de listes chaînées :

```
typedef struct _OneList OneList;
struct _OneList
{
        //Tete d'une chaînée de table de réimplantation
        ReimpTab r;
        //Nom de cette liste chaînée
        char name[26];
        //Table de réimplantation suivante
        OneList * suivant;
};
typedef struct
{
        //Tête de la liste chaînée
        OneList  * tete;
        //Nombre d'élément dans notre liste chaînée
        (Nombre de tables de réimplantations)
        int nb;
} ListReimpTab;
```

II – PHASE 1

Pour les étapes qui suivent nous avons utilisé comme doc le pdf elf.pdf fourni sur la page du projet et également de la doc sur internet

(source : « https://en.wikipedia.org/wiki/Executable_and_Linkable_Format »).

Nous avons également utilisé une autre doc pour pourvoir utiliser le 64bits

(source : « http://manpagesfr.free.fr/man/man5/elf.5.html »)

La description de chaque étapes ci dessus sera souvent composé de 2 paragraphes :

- Le premier paragraphe concernera le remplissage de notre structure Ofile qui sera toujours réaliser dans le fichier **read_func.c**.
- Le deuxième paragraphe concernera l'affichage de l'étape concerné qui sera toujours réaliser dans le fichier **print.c**.

Notre but pour cette partie est d'avoir le même affichage que la commande "readelf -a fichier.o" avec des adaptations par rapport aux docs.

1 – Affichage de l'entête

(function read_header)

Pour réaliser l'affichage de l'entête de notre fichier objet au format ELF nous nous somme basé sur la doc fourni (elf.pdf) afin de savoir combien d'octets nous devions lire pour chaque éléments (exemple en Annexe 1). Puis nous avons remplis la structure **Elf64_Ehdr** présente dans notre structure **Elf_Header**. Pour pouvoir lire l'entête complète cela n'est pas compliqué car nous commençons dès le début du fichier (préalablement ouvert) présent dans le descripteur de fichier se trouvant dans notre structure principal Ofile et remplissons les variables nécessaire par une succession de fread utilisé via les fonction "read_Elf…"

(function print_header)

Puis pour afficher dans la console le contenue de **Elf_Header** nous avons réaliser une succession de printf afin d'essayer de correspondre au mieux à l'affichage de la commande "readelf -h fichier.o".

2 – Affichage de la table des sections

(function read_section_headers)

Pour réaliser l'affichage de la table des sections de notre fichier objet nous avons répété la même opération que précédemment. Nous avons continuer la lecture via fread en se plaçant dans le fichier à l'offset **e_shoff** (variable définit précédemment dans le header) grâce à fseek et avons par la suite créés une boucle se répétant **e_shnum** fois (variable définit précédemment dans le header). Ces deux variable correspondent respectivement à l'endroit où se situe la section header ainsi que la taille de cette dernière. Cette section représente une table énumérant toute les sections existante dans le fichier. Nous avons donc remplis toutes les variables nécessaires de la structure **Elf64_Shdr** présente dans notre structure **SecHead**.

(function print_section_headers)

Puis pour afficher dans la console le contenue de **SecHead** nous avons réaliser une succession de printf afin d'essayer de correspondre au mieux à l'affichage de la commande "readelf -S fichier.o".

3 – Affichage du contenu d'une section

(function print_section)

Pour cette fonction on cherche une section (précisé en argument) dans la table des sections (grâce à notre liste chaînée) afin de récupérer l'offset et la size de cette dernière. Puis on se place à l'endroit voulu (sh_offset) via la fonction fseek et affichons son contenu dans la console comme précédemment grâce à une boucle for allant jusqu'à sa size (sh_size).

4 – Affichage de la table des symboles

4.1 – Table des symboles

(function read_table_symboles)

Pour cette fonction on cherche un type équivalent à SHT_SYMTAB dans la section header. Puis on se place à l'endroit voulu (sh_offset) via la fonction fseek et affichons son contenu dans la console comme précédemment grâce à une boucle for allant jusqu'à sa size (sh_size). C'est ainsi que nous remplissons notre structure **SymTab**.

(function print_table_symboles)

Puis pour afficher dans la console le contenue de **SymTab** nous avons réaliser une succession de printf afin d'essayer de correspondre au mieux à l'affichage de la commande "readelf -s fichier.o".

4.2 – Tables des strings

(function read_string_table)

Pour cette fonction on cherche un type équivalent à SHT_STRTAB dans la section header. Puis on récupère l'offset et la size. Puis on se place à l'endroit voulu (l'offset) via la fonction fseek et rajoutons chaque caractère dans notre structure grâce à une boucle for allant jusqu'à sa size. C'est ainsi que nous remplissons notre structure **StringTab**.

5 – Affichage de la table des réimplantations

(function read_table_reimplantation)

Pour cette fonction on cherche un type équivalent à SHT_REL ou SHT_RELA dans la section header. Puis on se place à l'endroit voulu (sh_offset) via la fonction fseek et affichons son contenu dans la console comme précédemment grâce à une boucle for allant jusqu'à sa size (sh_size). C'est ainsi que nous remplissons notre structure **ReimpTab**.

(function read_table_reimplantation_new)

Cette fonction appel la fonction précédente le nombre de fois qu'il y a d'élément SHT_REL et donne en argument quel élément il faut selectionner.

(function print_table_reimp)

Puis pour afficher dans la console le contenue de **ReimpTab** (passé en argument) nous avons réaliser une succession de printf.

(function print_table_reimp_new)

Cette fonction appel la fonction précédente autant de fois qu'il y a de tables de réimplantations, avec à chaque fois la table en question afin de l'afficher. Nous cherchons à nous rapprocher au mieux de l'affichage produit par la commande "readelf -r fichier.o"

III – PHASE 2

> Nous nous somme pas occupé du changement de l'offset à partir de la partie 7...

1 – Production d'un fichier résultat au format ELF

Pour réaliser la fusion, nous avons tout d'abord récupéré les 2 fichiers ELF (1 et 2) via les arguments de la ligne de commande. Nous avons décidé de fusionner les deux fichiers tout en produisant simultanément le nouveau fichier de sortie au format ELF. On écrit alors le contenu des fichiers fusionnés dans l'ordre indiqué dans la documentation ELF fournie, puis on modifie les octets qui change lors de la fusion.

(function fusion)

Dans un premier temps, on appelle cette fonction qui va réaliser toutes les étapes de la fusion des deux fichiers en paramètre.
On commence par copier le header du premier fichier ELF dans le fichier de destination, puis on l'écrit dans le fichier de sortie à l'aide de la fonction write_header().

2 – Fusion, renumérotation et corrections des symboles

(function fusion_table_symbole)

On copie la table des symboles (1 et 2) des deux fichiers à fusionner, puis on récupère la référence du dernier symbole de la table 1 (liste chaînée). On va ensuite parcourir toute la seconde table des symboles qu'on va ajouter à la première table. Pour chaque symbole de la table 2, on va appeler la fonction (comparaison()) qui vérifie comment le symbole peut être ajouté à la première table. Une fois la première table des symboles complètement fusionnée, on modifie le nombre de symboles de cette table dans le header de la section correspondant à cette table (sh_info).

(function fusion_reimp)

Cette fonction va ajouter toutes les réimplémentations d'une table B dans une table A (dans la struct **ReimpTab**). On récupère la référence de la derniere réimplantation dans A. On crée ensuite une nouvelle table (**OneReimp**) que l'on va remplir avec le contenu de B. On finit par l'ajouter à la suite de A.

(function comparaison)

Si le symbole testé est local, on l'ajoute directement à la suite de la référence du dernier élément de la table des symboles 1 (fonction précédente).

Si le symbole est global, on doit alors parcourir tous les symboles de la tables des symboles 1.

Si le symbole ne correspond pas à un autre, on peut l'ajouter à la suite.

Sinon, si on trouve un symbole qui possède le même nom que celui testé :

– Si les deux symboles sont définis, on termine le programme de fusion (erreur).

– Si les deux symboles ne sont pas définis, on laisse le symbole déjà présent dans la table 1 et on ajoute pas de nouveau symbole.

– Si le symbole est défini dans la table 1 mais pas dans la table 2, alors on ne fait rien.

Si le symbole est défini dans la table 2 mais pas dans la table 1, on remplace le symbole de la table 1 par le symbole de la table 2.

3 – Fusion et corrections des tables des réimplantations

(function fusion_table_reimplementation)

Pour cette fonction, on a commencé par copier la liste des tables de réimplantation des deux fichiers. On parcourt ensuite toutes les tables du fichier 1 et pour chacune de ces tables, on cherche une table identique dans le fichier 2 avec laquelle on va pouvoir fusionner à l'aide de la fonction (fusion_reimp()). On garde également en mémoire toutes les tables du fichier 2 qui on pu fusionner avec une table de nom identique dans le fichier 1 afin de connaître à la fin toutes les tables du fichier 2 qui ne sont pas dans le fichier 1.

C'est ainsi qu'on ajoute toutes ces tables restantes à l'aide de la fonction (copie_reimp)

(function copie_reimp)

Cette fonction va ajouter à la suite d'une liste de table de réimplantation A une table de reimplantation T contenu dans une liste de table de réimplantation B. On passe en paramètre le numéro de la table (correspondant à une table non fusionnée avec fusion_réimp() dans la fonction fusion_table_réimplantation()). On commence par récupérer la référence de la dernière table dans A. Puis la référence de la table T dans B à ajouter avec le numéro en paramètre.
On copie ensuite cette table à la suite de A : On crée une nouvelle table de réimplantation qu'on alloue puis remplie avec le contenue de T en parcourant tous les éléments de cette table de réimplantation.

4 – Fusion et renumérotation des sections

(suite fonction fusion)

On commence par parcourir toute les sections header du fichier 1. On récupère l'offset de sa position actuelle dans le fichier 1, puis on l'écrit dans le fichier de destination à l'aide de la fonction write_section(), qui écrit le contenu d'une section dans le fichier de destination à l'aide du header de cette section.
On cherche ensuite dans le fichier 2 si une section identique existe, auquel cas on ajoute à la suite du fichier de destination son contenu, tout en modifiant le nouveau offset que va avoir la prochaine section après l'ajout de ces nouveaux octets, ainsi que le header de cette section qui sera modifié (taille).

On écrit ensuite les en-têtes des sections à la suite du fichier de destination.

(function write_section_header)

Cette fonction parcourt chaque section header du OFile de destination précédemment fusionné et écrit à l'aide d'une succession de write_word() tout le contenu d'une section header ordonnée comme la documentation l'indique.

5 – Correction

On termine la fusion par corriger dans l'entête du fichier ELF de destination les modifications apportées pendant la fusion (**e_shoff :** Le offset des en-têtes des sections). à l'aide d'un fseek et du offset calculé pendant la fusion des sections, on se place au champs correspondant et on le modifie avec un write_word().

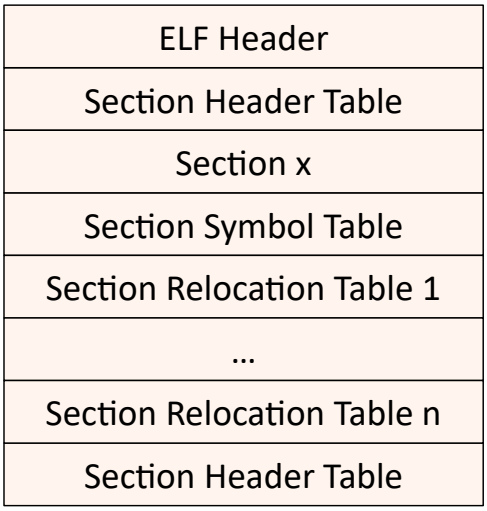
Le fichier de destination est maintenant la fusion entre les deux fichiers donnés en entrée.

IV – NOTRE ORGANISATION

1 – Fonctionnalité implémenté

Toute les fonctions demandés sont présentes. Ces dernières sont fonctionnelles à l'exception de la partie 8 qui nous donne un résultat incohérent si les noms des tables de réimplantations des deux fichier sont de même noms… En revanche si les noms sont différents tout ce passe très bien et fonctionne.

• La première phase (afficher le contenu d'un fichier objet ELF, cf. ci-après)



Avec "ELF Header" qui correspond à la partie 1 de la phase 1.

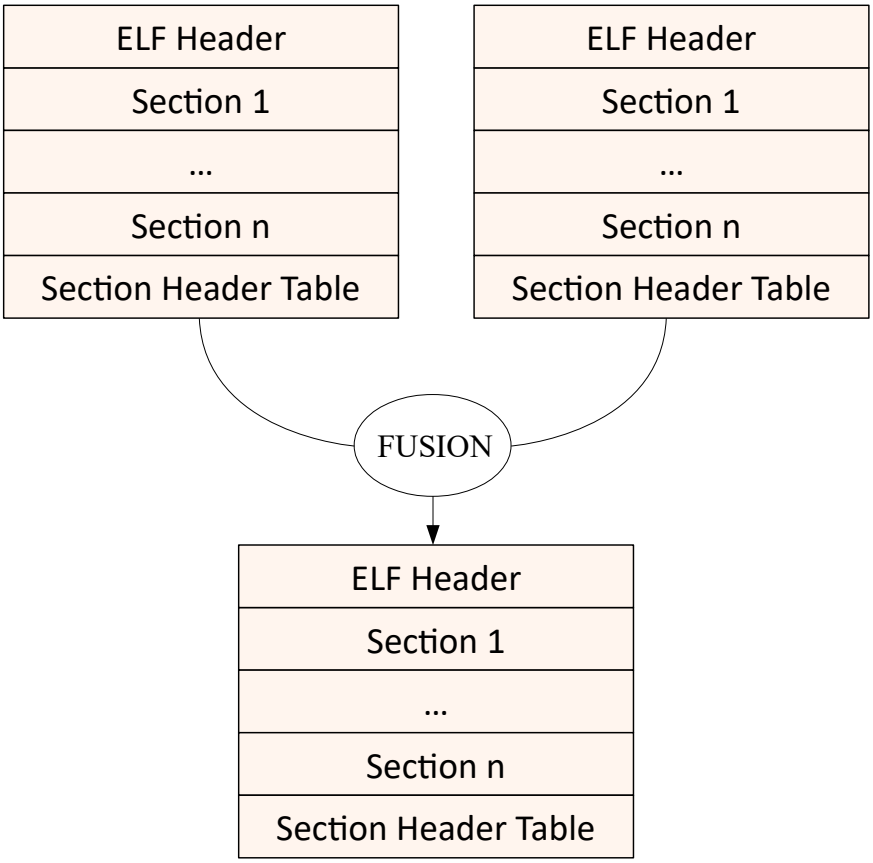
Avec "Section Header Table " qui correspond à la partie 2 de la phase 1.

Avec "Section x" qui correspond à la partie 3 de la phase 1.

Avec "Section Symbols Table" qui correspond à la partie 4 de la phase 1.

Avec "Section Relocation Table 1…n" qui correspond à la partie 5 de la phase 1.

• La deuxième phase (fusionner deux fichiers objets ELF en un seul, cf. ci-après)



2 – Bogues non résolus

Nous avons deux bogues qui se trouve pour le premier être un bogue d'affichage. En effet lors de l'exécution de la partie1 sur un fichier objet, dans la section table header, les noms des sections sont plus ou moins long, ce qui a pour conséquence de décaler l'affichage à certains moments. Les numéros correspondant à un entête précise se voit décaler de une voir plusieurs colonne, ce qui rend la lecture compliquée... Pour remédier à ça il faudrait calculer la taille du nom et agir en fonction de ce dernier afin de pouvoir réaligner toute les infos en face des bonnes colonnes.

Notre deuxième bogue est quand a lui une erreur lors de la fusion au niveau des tables de réimplantations. Si lors de la fusion les tables de réimplantations des deux fichiers ont le même noms alors des problèmes apparaîtrons lors de la vérification (cf. test n°7 ci-après).

3 – Nos tests effectués

3.1 – Objectif

Test n°	Objectif	Partie
1	affiché le ELF header	I
2	affiché la section table header	I
3	affiché le contenue de la section 3	I
4	affiché la table symbole	I
5	affiché la section de réimplantation	I
6	format 64/32 bits	I
7	vérification de fusion	II

3.2 – Marche à suivre

Test n°	Méthode	Partie
1	Nous avons utilisé "readelf -h EX/example1.o" et "./partie1 EX/example1.o 3" pour comparer les deux header afin de savoir les erreurs possible.	I
2	Nous avons utilisé "readelf -S EX/example1.o" et "./partie1 EX/example1.o 3" pour comparer les deux section table header afin de savoir les erreurs possible.	I
3	Nous avons utilisé "./partie1 EX/example1.o" afin de voir le nombre d'octets présent dans la section 3 et avons vérifié si cela coïncidait avec sa size donné.	I
4	Nous avons utilisé "readelf -s EX/example1.o" et "./partie1 EX/example1.o 3" pour comparer les deux section table symboles afin de savoir les erreurs possible.	I
5	Nous avons utilisé "readelf -r EX/example1.o" et "./partie1 EX/example1.o 3" pour comparer les deux listes de section tables réimplantation afin de savoir les erreurs possible.	I
6	Nous avons utilisé les 5 mêmes type de test que précédemment mais avec la structure voulu (32bits ou 64bits)	I
7	Pour vérifier chaque partie de la fusion nous utilisions l'exécutable partie1 afin de vérifier si les valeurs sont cohérentes où non.	II

3.3 – Conclusion du résultat

Test n°	Conclusion	Partie
1	Nous avons des résultats identiques de ceux de la commande readelf à l'exception de certains noms qui diffèrent car nous n'avons pas toute la doc nécessaire pour implémenter cela. (ex : 0 → SHN_UNDEF, 0xff00 → SHN_LOPROC, 0xfff1 → SHN_ABS etc. …). Donc lorsque nous n'avons pas le noms de la constante nous mettons uniquement sa valeur.	I
2	Même chose que pour le test n°1	I
3	Le nombre d'octets affichés correspond bien au nombre d'octets de la size précisé.	I
4	Même chose que pour le test n°1	I
5	Même chose que pour le test n°1	I
6	Même chose que pour le test n°1	I
7	Les valeurs retrouvés suite à l'exécution de la partie1 sur la fusion semblent cohérents.	II

4 – Répartition du temps de travail *(cf. Annexe 2)*

V – ANNEXES

1 – Annexe 1

```
#define EI_NIDENT 16
typedef struct {
        unsigned char e_ident[EI_NIDENT];
        Elf32_Half e_type;
        Elf32_Half e_machine;
        Elf32_Word e_version;
        Elf64_Addr e_entry;
        Elf64_Off e_phoff;
        Elf64_Off e_shoff;
        Elf32_Word e_flags;
        Elf32_Half e_ehsize;
        Elf32_Half e_phentsize;
        Elf32_Half e_phnum;
        Elf32_Half e_shentsize;
        Elf32_Half e_shnum;
        Elf32_Half e_shstrndx;
} Elf64_Ehdr;
```

2 – Annexe 2

T : Thomas | P : Paul | D : Dorian | N : Nicolas | L : Loris | M : Marie

☒ : Étape fini