PROG 5 Projet 2021-2022

Descriptif de la Structure du Code Dévéloppé

Kimberly Beauvais, Xuan Li, Nathaniel Tobing
Hugo Roger, Emilien Maillard-Simon, Théo Lanneau

13 janvier 2022

Table des Matières

Principe du Projet

L'objectif de ce projet est d'implémenter une sous partie d'un éditeur de liens. Plus précisément, le projet est centré sur la dernière phase, dite de réimplantation, exécut´ee par l'éditeur de liens. Ce document contient une description générale du code créé tout au long du projet, qui est divisé en deux phases : Fusion et Implantation

1 Phase 1 : Fusion

La première phase du projet consiste à rassembler les différentes zones (sections) définis dans les fichiers objets donnés en entrée. Le programme principal, *./readelf*, affiche des informations sur un objet au format ELF. Les options contrôlent les informations particulières à afficher. *elffile* ... sont les fichiers objets à examiner. Les fichiers ELF 32 bits sont supportés, tout comme les archives contenant des fichiers ELF sont supportés. Ce programme exécute une fonction similaire à *objdump* et *readelf* mais avec seulement les fonctionnalités spécifiées dans la sous-section options.

Chaque étape est divisée en deux parties, la première qui récupère les données du fichier et la seconde qui traite les données et imprime le résultat.

1.1 elf_header

Ce fichier contient les fonctions permettant d'afficher les informations contenues dans l'en-tête ELF au début du fichier.

Obtenir l'en-tête du fichier

Afin d'obtenir l'en-tête du fichier, on lis les 16 premiers bytes dans *elffile* et le stocker dans le structure : *filedata->header.e_ident*. Après avoir vérifié que le fichier est bien un fichier ELF, on lis le reste de l'en-tête du fichier, effectuons l'opération big endian sur chaque structure et le stocker dans *filedata->header*.

Traitement de l'en-tête du fichier

Pour afficher les données dans *filedata->header*, on a crée des sous fonctions pour obtenir le bon nom selon les valeurs obtenu depuis *get_file_header*.

1.2 elf_shdrs

Ce fichier contient les fonctions permettant d'afficher les informations contenues dans les en-têtes de section du fichier, si celui-ci en possède.

Obtenir la table des sections

Afin d'obtenir la table des sections, on verifie qu'il existe des entrées dans le tableau depuis les données dans *filedata->header*. S'il y en a, on récupére la table des sections à partir du *elffile* et pour chaque section, on effectue le fonction big_endian et stocker le resultat dans *filedata->section_headers*. Dans cette fonction, on obtient également les données pour le tableau des chaînes de caractères pour obtenir le nom des sections.

Traitement de la table des sections

Pour traiter les données dans *filedata->section_headers*, on a crée des sous fonctions pour obtenir le bon nom selon les valeurs obtenu depuis *get_section_headers*. Ensuite, on affiche le clé des flags.

1.3 read_section

Ce fichier contient les fonctions permettant d'afficher le section saisie s'il existe dans *elffile*.

La fonction *read_section* permet de lire le contenu brut (sous forme héxadécimale) d'une section saisie par l'utilisateur. Elle trouve l'emplacement de la section correspondante en comparant son nom avec ceux qui sont dans la section header et après, affiche le contenu dans terminal.

1.4 elf_symbol_table

Ce fichier contient les fonctions permettant d'afficher le table des symboles.

Obtenir la table des symboles

Afin d'obtenir la table de symboles, on récupère le string table depuis *elffile*. Ensuite, utilisant *section_headers[index_symtab].sh_offset* dans l'en tête des sections de la table de symboles, on récupère toutes les entrées.

Traitement de la table des symboles

Pour traiter les données dans *filedata->symbol_table*, on a crée des sous fonctions pour obtenir le bon nom ou valeur selon les données obtenu depuis *get_symbol_table*.

1.5 elf_reltab

Ce fichier contient les fonctions permettant d'afficher les tables de reimplementation.

Obtenir les tables de reimplementation

Afin d'obtenir les tables de reimplementation du fichier, on commence par compter le nombre de sections de relocalisation qui se trouvent dans *elffile* pour alluer le bon taille de memoire à *filedata->reloc_table.rel_tab*. Ensuite, si une section a le type REL, alors on ajoute les entrées de cette section dans le tableau de réimplantation.

Traitement de les tables de reimplementation

Pour traiter les données dans *filedata->reloc_table*, on transform les bits des données obtenu depuis *get_rel_table* utilisant un fonction big endian.

1.6 elf_main

Ce fichier contient les fonctions pour obtenir le data et les processer selon les options donnée pour l'utilisateur. Le but de ce fichier est de rassembler toutes les fonctions de la phase 1 pour obtenir un fichier de sortie homogène mais aussi d'obtenir un structure data utilisable pour la phase 2.

Structures de Données

On a créé une nouvelle structure de données *Filedata* qui contient toutes les données nécessaires du fichier qui seront manipulées dans les phases 1 et 2.

```
typedef struct filedata {
    const char *        file_name;          // file name
    Elf32_Ehdr          file_header;        // file header
    FILE *              file;               // pointer to file
    uint32_t            file_offset;        // offset of file
    uint32_t            file_size;          // size of file
    Elf32_Shdr *        section_headers;    // section header table
    Elf32_Rel_Tab       reloc_table;        // relocation table
    Elf32_Sym_Tab       symbol_table;       // symbol table
    char *              string_table;       // string table
    int                 string_table_length;// string table size
} Filedata;
```

On a créé une nouvelle structure de données *Elf32_Ext_Rel* qui contient la table des relocation et quelques variables qui peut-être utile.

```
typedef struct{
    Elf32_Word  rel_sh_name;    //name of the relocation table
    section
    Elf32_Off   rel_sh_offset;  //offset of the section
    Elf32_Half  rel_ent_num;    //number of Elf32_Rel entries
    Elf32_Rel*  rel_ents;       //table of entries
} Elf32_Ext_Rel ;
```

On a créé une nouvelle structure de données *Elf32_Rel_Tab ;* qui contient le nombre de table de relocation et une table de type *Elf32_Ext_Rel*.

```
typedef struct{
    Elf32_Half      rel_num; //number of relocation table sections
    Elf32_Ext_Rel*  rel_tab; //table of relocation tables
    pertaining to those sections
} Elf32_Rel_Tab;
```

On a créé une nouvelle structure de données *Elf32_Sym_Tab ;* qui contient le nombre de symboles et la table de symbol elle-même.

```
typedef struct{
    Elf32_Half sym_tab_num; //number of symbol table entries
    Elf32_Sym* sym_entries; //actual symbol table
} Elf32_Sym_Tab;
```

Big Endian

Les fonctions big endians sont utilisé pour transoformer les bytes obtenu depuis le fichier objet en big endian. *field* est le bit qu'on veut transformer. Pour n'importe quel type de donnée, le concept reste le même :

```
1     1 bit  : return field
2     2 bits : return (field[1] | field[0] << 8)
3     3 bits : return (field[2] | field[1] << 8 | field[0] << 16);
4     4 bits : return (field[3] | field[2] << 8 | field[1] << 16 |
5                      field[0] << 24);
```

Options

Dans le programme principal pour la phase 1, on a ajouté des options d'affichage selon les besoins d'utilisiateur. Les options sont les suivant :

-a	Equivalent à : -h -S -s
-e	Equivalent à : -h -S
-h	Affichage de l'en-tête de fichier ELF
-S	Affichage de la table des sections
-s	Affichage de la table des symboles
-x	Affichage du contenu d'une section
-r	Affichage des tables de réimplantation

Obtenir les Données du fichier ELF

Le but de la fonction *get_filedata* est d'obtenir tout le données du structure *filedata* depuis la fichier d'objet donnée. Ce fonction va être utilisé aussi pour phase 2.

1.7 readelf

Ce fichier contient le programme principal de la phase 1 de ce projet. Il était créer pour séparer les deux phases tout en réutilisant les fonctions créées en phase 1 dans la phase 2.

2 Phase 2

La deuxiéme phase du projet consiste à modifier le contenu du *elffile* afin d'effectuer l'implantation. Le programme principal prend en paramètre les adresses auxquelles les sections du programme (typiquement .text et .data) doivent être chargées. Le résultat du programme est mis dans fichier nommé *out.bin*.

2.1 relocation

Ce fichier contient le programme principale de la phase 2 et les fonctions pour les étapes 6 (Renumérotation des sections) et 7 (Correction des symboles).

Renumérotation des sections et Correction des symboles

Le but de la fonction *renumerotation* est de renuméroter toutes les sections une fois que les sections de réimplantation ont été supprimé de la table. Pour le faire, on commence par copier les données nécessaires du *filedata* dans un nouveau structure *(newfiledata)*. Ensuite, la table des sections est obtenu en vérifiant que la type de section n'est pas un réimplantation avant de le stocker dans *(newfiledata)*.

Afin d'obtenir l'index correct de la table des symboles, à l'aide d'une boucle for, on parcours la table des symboles pour renumére les sections et modifie la valeur correspondante.

Création d'un fichier binaire

Le but de la fonction *write_file* est de généré un fichier binaire en copiant les données utilie de la fichier originale dans le nouveau fichier (*out.bin*).

2.2 réimplantation_type

Ce fichier contient les fonctions de réimplantation selon le type dans le *r_info* de la table de réimplantation.

réimplantation

Le but de la fonction *implantation* est de mettre la nouvelle valeur de symbol calculer en utilisant la fonction *calcul_val* dans le bon endroit selon les tables de réimplantation.

Différence entre type R_ARM_ABS* et R_ARM_JUMP24/R_ARM_CALL

La différence entre R_ARM_ABS* et R_ARM_JUMP24/R_ARM_CALL est le moyen de calculer le valeur de symbol. Pour R_ARM_ABS*, on obtient le nouveau valeur de symbol directement depuis la nouvelle table de symboles. Pour R_ARM_JUMP24/R_ARM_CALL, on prend le nouveau valeur de symbol et applique le masque spécifié dans le documentation de ELF pour ARM.

2.3 out.bin

C'est le fichier binaire produit par la phase 2. Il contient les données pour le *elffile* après avoir fait les réimplantation.