UFR IM²AG
UNIVERSITÉ Grenoble Alpes

Année universitaire 2021-2022

- Rapport de Projet logiciel - Chargement d'un exécutable au format ELF

Groupe 12

Projet réalisé par :

Bounhach Walid
Pereira Mickael
Berruyer William
Houny Julien
Ducros Arthur
Ben Youssef Rania

Projet encadré par :

Guillaume Huard
Vincent Danjean
Philippe Waille
Renaud Lachaize
Anne Rasse
Cyril Labbé

Remerciements

Nous tenons, au nom du groupe, à remercier et à témoigner notre reconnaissance à tous les responsables avec qui nous avons communiqué via la hotline pour leur soutien lors des séances réalisées tout au long du projet.

SOMMAIRE

1. Introduction

Ce rapport présente le chargement d'un exécutable en format ELF effectué lors du projet de fin de semestre de l'année universitaire 2021-2022. Nous y retrouverons les explications nécessaires à la compréhension du code fournis ainsi que les détails de compilation et de lancement du projet. Sans oublier les diverses listes demandées concernant les fonctionnalités implémentées, les bogues connus et les tests effectués.

2. Compilation & Lancement

Une fois le code télécharger, nous devons nous placer dans le répertoire principal qui fait office de racine du projet. Une fois dans ce répertoire, il suffira de suivre les consignes suivantes.

2.1. Compilation

Pour compiler notre programme, il suffira tout simplement d'utiliser la commande suivante :

```
> make
```

Notre Makefile se chargera de compiler notre code et ainsi générer les fichiers exécutables suivant :

- read_elf
- renum

Les étapes 1 à 5 sont regroupées dans l'exécutable « read_elf ».

L'étape 6 est donc regroupée dans l'exécutable « renum »

2.2. Lancement de l'exécutable

- Pour lancer l'exécutable *read_elf*, nous aurons d'abord besoin de connaître les options disponibles.

Nous avons l'option :

- -h : Affiche l'en-tête d'un fichier ELF.
- -S : Affiche l'en-tête des sections.
- -S -rel : permet d'afficher les sections sans les relocations.
- -x <string> : Affiche le contenu de la section avec le nom <string> en hexadécimal.
- -r : Affiche les réadressages si ceux-ci sont présents.
- -s : Affiche la table des symboles.

Voici maintenant quelques exemples d'exécution de l'exécutable :

```
> ./read_elf -S elf_linker-1.0/Examples_loader/example1
```

Ici, nous allons afficher l'en-tête des sections du fichier example1.

```
./read_elf -x 14 elf_linker-1.0/Examples_loader/example3
```

Cette commande va permettre d'afficher le contenue de la section qui a pour nom 14.

- Pour lancer l'exécutable *renum*, il suffira d'écrire :

```
./renum <votreExemple> <fichierDeSortieQuiVaEtreCree>
```

Ceci est censé supprimer les sections de type « *rel* » de notre exemple.

2.3. Lancement des jeux de tests

Pour les jeux de tests, nous utiliserons en majorité ceux fournis par les enseignants. Essentiellement, parce que ces derniers sont compilés en ARM.
Nous avons adaptés au début notre code sur du Intel mais le projet mettait bien en évidence que notre programme devait être réaliser sur du ARM. Les jeux de tests se trouvent donc tous dans le dossier examples_loader.

```
elf_linker-1.0/Examples_loader/example*
```

3. Structure du code & des fichiers

Voici une liste des fichiers permettant le bon fonctionnement de notre projet :

- *entete_elf.c* : représente le programme principal, il regroupe toutes les options que nous avons implémentées dans ce projet. C'est ici que nous gérons si le fichier est en 32 bit ou 64 bits. Si nous affichons un message d'erreur où non pour certains fichiers.
- *Affiche_entete_elf.c* : permet d'afficher l'en-tête d'un fichier ELF. Nous utiliserons des « « *switch case* » sur les différents champs du Elf32_Ehdr.
- *converter.c* : sert à convertir les bits Big Endian en Little Endian. Les fonctions implémentées dans ce fichier sont adaptées pour les pointeurs de nos structures.
- *table_section_elf.c* : implémente la table des sections d'un fichier ELF.. Nous permettra d'avoir un rendu similaire à la commande « *readelf* » sur les sections. Et ainsi, obtenir les différents flags, noms, offset des sections du programme.
- *table_symbole.c* : implémente la table des symboles d'un fichier ELF. Ce fichier est implémenté différemment des autres fichiers. Notamment parce que nous utilisons une projection de la mémoire avec *mmap*. Nous obtenons donc, à la fin, les symboles et des informations comme avec la commande « *readelf* » sur ces derniers.
- *read_section_elf.c* : permet de faire la lecture des sections d'un fichier ELF donné en argument. C'est ici que nous réalisons une traduction hexadécimale vers ASCII.
- *table_relocation_elf.c* : implémente la table de réimplantation. Les fonctions sont réalisées en 32 et 64 bits. Nous les avons testées sur un fichier 64 bit qui contenait des réimplantations. Il est réalisé de la même manière que table_symbole avec une projection via *mmap*.
- *renumerotation.c* : permet d'effectuer la renumérotation des sections d'un fichier ELF. Ici, nous essayons de supprimer la section des rimplantation via deux projections de mémoire, une pour le fichier d'entrée et une autre pour le fichier de sortie afin de conserver l'état du fichier d'entrée. Le fichier de sortie sera écrit via *memcpy*.
- *test.sh* : un script shell qui va pouvoir faire des tests de comparaisons avec notre programme et la commande « *readelf* » (options comprises).

4. Fonctionnalités

4.1. Fonctionnalités implémentées

Toutes les étapes de la phase 1 ont été implémentées dans se projet.

4.1.1. Affichage de l'en-tête

L'utilisation des fichiers suivants sont nécessaires afin d'avoir l'affichage de l'entête d'un fichier ELF.

- entete_elf.c (programme principal)
- Affiche_entete_elf.c
- converter.c

L'affichage de l'en-tête est donc pleinement implémenté et suit la spécification jusqu'au bout. L'étape 1 est donc validée sans problème.

4.1.2. Affichage de la table des sections

Nous allons utiliser les fichiers listés ci-dessous afin de pouvoir afficher la table des sections qui correspond à la commande *readelf* avec l'option *-S*.

- entete_elf.c (programme principal)
- table_section_elf.c
- converter.c

L'étape 2 est réalisée avec succès.

4.1.3. Affichage du contenu d'une section

L'affichage du contenu d'une section, ici, va avoir besoin des fichiers :

- entete_elf.c
- table_section_elf.c
- read_section_elf.c
- converter.c

L'affichage demandé dans le sujet a été respecté, notre fonction principale agit comme la commande *readelf -x <string>*. L'étape 3 est elle aussi validée.

4.1.4. Affichage de la table des symboles

Pour pouvoir afficher la table des symboles, il va nous falloir utiliser :

- entete_elf.c
- table_symbole.c
- converter.c

L'affichage de la commande *readelf -s* correspond parfaitement avec l'affichage que nous obtenons avec notre code.
L'étape 4 est donc validée.

4.1.5. Affichage des tables de réimplantation

L'affichage de réimplantation aura besoin de ces fichiers pour pouvoir bien fonctionner :

- entete_elf.c
- table_relocation_elf.c
- converter.c

En utilisant la commande précisée dans le sujet du projet *readelf -r*, et en la comparant avec le résultat de notre programme, nous arrivons à avoir le même affichage.
L'étape 5 est validée.

4.1.6. Renumérotation des sections

La renumérotation des sections va prendre en compte les fichiers suivants :

- renumerotation.c

Cette partie est indépendante des autres. Elle représente la suppression des sections de type « rel ». Nous procédons via des *mmap* et des descripteurs de fichiers. Nous rencontrons un problème lors de la synchronisation de notre *mmap* via la fonction *msync*.

4.2. Fonctionnalités non implémentées

Les étapes de 7 à 11 n'ont malheureusement pas été implémentées. Nous aurons donc les étapes de 1 à 5 qui fonctionneront correctement avec l'étape 6 qui quant à elle, générera des bogues.

5. Bogues connus

Voici la liste avec les détails des bogues connus dans notre projet :

- Nous ne gérons pas les fichiers x32 autre que du ARM comme spécifié sur le sujet.
- Nous ne gérons pas les fichiers x64 Tous processeurs / Hormis pour le redressage.
- N'affiche pas le symbole au niveau de l'étape 5.
- L'étape 6 ne fait pas ce que nous lui demandons. Le problème concerne la *mmap* qui ne se synchronise pas puisque *msync* nous retourne une erreur.

6. Script Shell

Nous avons un script shell nommé test.sh qui va nous permettre d'exécuter notre programme principal avec les options que nous avons implémenté tout en comparant l'affichage obtenu avec les véritables commandes mentionnées dans le sujet du projet.

> ./test <option> elf_linker-1.0/Examples_loader/example1

Vous pouvez voir ci-dessous, un exemple de son exécution :

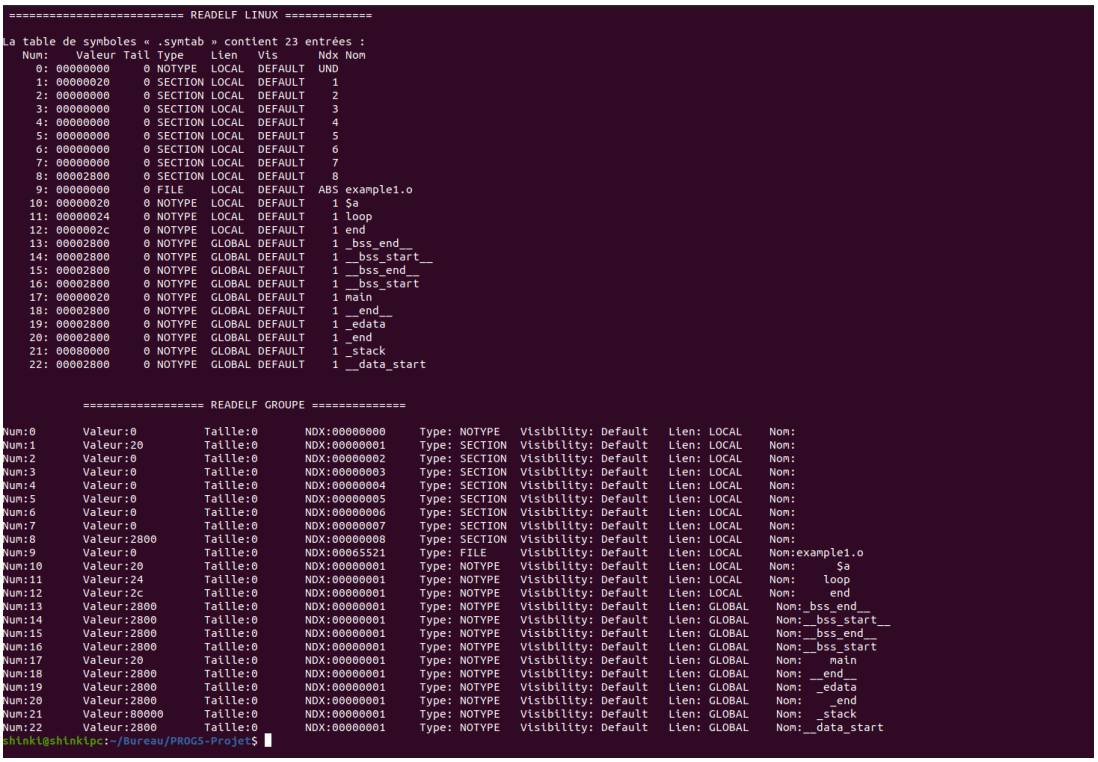

Image 1- Exemple de l'exécution du script shell

7. Journal du projet

En ce qui concerne le journal du projet, nous vous avons mis un document en format pdf afin que vous puissiez suivre l'évolution du projet tout au long des deux semaines. Ce fichier se nomme donc *Journal de projet .pdf.*