GROUPE 9

REALISATION D'UN EDITEUR DE LIENS

PROG5 - Projet, année 2021-2022

AVOGNON Laëtitia – BULTINCK Léonard – CLÉMENT Tom –
FIORI Victoria – GÖRÜR Selim – MARCHIONINI David
13/01/2022

Table des matières

Mode d'emploi 2
Compiler le projet 2
Lecture d'un fichier ELF 2
Réimplantation d'un fichier ELF 2
Structure du code développé 3
Organisation générale du code 3
Structures et macros 3
Fonctions 4
Lecture 4
Affichage 4
Fonctions utiles 4
Réimplantation 4
Tests 4
Choix des tests 4
Automatisation des tests 5
Fonctionnalités implémentées 6
| Première phase 6
ÉTAPE 1 6
ÉTAPE 2 6
ÉTAPE 3 6
ÉTAPE 4 6
ÉTAPE 5 6
| Seconde phase 7
ÉTAPE 6 7
ÉTAPE 7 A 11 7
Liste des bogues connus 7
Tests 8
| Première phase 8
ÉTAPE 1 8
ÉTAPE 2 8
ÉTAPE 3 8
ÉTAPE 4 8
ÉTAPE 5 9
| Seconde phase 9
ÉTAPE 6 à 11 9

Mode d'emploi

Compiler le projet

Pour compiler le projet, placez-vous à la racine du projet.

Deux exécutables se trouveront à la racine : *read_elf* et *reloc_elf*.

```
$       . /configure
$       make
```

Pour les tests automatisés, les scripts Bash suivant sont à la racine du projet :

- *test_header.sh* : permet de tester la lecture des en-têtes
- *test_sect_head.sh* : permet de tester la lecture de la table des sections
- *test_contenu_section.sh* : permet de tester la lecture du contenu des sections
- *test_table_symbole.sh* : permet de tester la lecture de la table des symboles
- *test_reloc.sh* : permet de tester la lecture de la table des réimplantations
  (Voir la Tests pour + d'informations)

Lecture d'un fichier ELF

Pour lire un fichier ELF, la commande à utiliser est celle-ci:

```
$       . /read_elf [ --help ] [ -h | -S | -s | -x <num|text> | -r ] FICHIER
```

Le programme a le même comportement que le *readelf* standard et affichera les différentes informations souhaitées selon les options.

Options :
-h : En-tête ;
-S : Table des sections ;
-s : Table des symboles ;
-x <num|text> : Affichage du contenu d'une section en hexadécimal uniquement ;
-r : Table des réimplantations

Réimplantation d'un fichier ELF

Pour réimplanter un fichier ELF, la commande à utiliser est celle-ci :

```
$       . /reloc_elf [ -H | --help ] [ -r ] FICHIER
```

Structure du code développé

Organisation générale du code

Pour la première phase nous avons décidé de créer une structure par étape, puisque les étapes contiennent des informations différentes en fonction de l'option utilisée.

Les structures et les macros sont dans le fichier *elf_lib.h*, une séparation en plusieurs modules par structures auraient pu être judicieuse mais les données utilisées pour ce projet sont peu nombreuses et la séparation modulaire aurait alourdie la structure et la gestion du code.

Cependant afin d'éviter de polluer les fonctions liées à la lecture et l'affichage de fichiers ELF, nous avons séparé les fonctions qui ne sont pas directement liés aux structures.

Le *main*, lui, se trouve dans un autre dossier : *elf_read*.

Chaque structure est initialisée à l'aide d'une fonction qui lira le fichiers ELF (dans le cadre du sujet, on se limite à la lecture de fichier ELF 32 bits) et sera ensuite affichée sur la sortie standard sous une forme lisible par l'humain à l'instar de la fonction *readelf* donné dans le sujet.

Le *main* permets de sélectionner les parties de codes à effectuer.

Pour la seconde phase, nous avons décidé de créer un deuxième programme, *elf_reloc*, qui réalise les différentes étapes de réimplantation. Il réutilise les structures et fonctions définies dans *elf_lib* de la première phase.

Structures et macros

Les structures sont similaires à celle de la documentation fournie.

Ces structures contiennent donc les informations brutes et non formatées du fichier. Les nombreuses macros permettent d'améliorer la lisibilité du code et sont, elles aussi, extraites de la documentation.

Nous avons choisi de passer par des structures car cela rend la programmation plus simple et plus facile à maintenir. Les données doivent être lues puis stockées afin de permettre aux fonctions de les réutiliser rapidement et efficacement.

Nous aurions pu utiliser directement le fichier en-tête *elf.h* qui est déjà présent en C.

Il y a 4 structures :

- Elf32 qui contient les informations liées à l'entête de fichier.
- Elf32_SH qui contient les informations d'un entête de section.
- Elf32_sym qui contient la table des symboles.
- Elf32_Rel qui contient les réimplantations.

Fonctions

Les fonctions sont divisées en 3 catégories celles de lecture du fichier, d'affichage sur la sortie standard et les fonctions utiles.

Lecture

Les fonctions de lectures lisent les données par bloc de la taille de la donnée à saisir. Par exemple un élément de taille 32bits sera lu entièrement et par pas de 4 bits. Puisqu'ici les données sont toujours de taille standardisée et ne peuvent pas variés.

Affichage

Les fonctions d'affichage regardent, pour chaque élément d'une structure, sa valeur hexadécimale. À l'aide de conditions, elles déterminent l'information formatée à afficher pour l'humain.

Fonctions utiles

Se situent dans *elf_utils.c* . Le fichier contient des fonctions auxiliaires comme bread qui permet de lire automatiquement en *little endian* ou bien en *big endian*. Mais aussi des fonctions pour lire les noms des sections/symboles depuis leur String Table respectifs.

Réimplantation

Les fonctions de réimplantation se trouvent dans le fichier *elf_reloc.c*. Elles vont permettre de renuméroter les sections sans compter les sections contenant des tables de réimplantation.

Tests

Choix des tests

Pour commencer à avancer, nous avons décidé de nous focaliser sur les exemples fournis par le sujet.
Afin de valider nos algorithmes par des tests, nous devons les confronter au maximum de cas possibles et, pour chacun, dans une quantité satisfaisante. À présent, nous avons 9 fichiers ELF créés lors de la compilation pour faire ces tests.

Automatisation des tests

Les tests ont été automatisés à l'aide de scripts bash. Il y a un script de test par partie décrite dans le sujet du projet, et lors de l'exécution d'un script, tous les fichiers dans le dossier de test seront testés.

Chaque test sauvegarde la sortie standard de notre programme dans un fichier et la sortie standard de readelf dans un autre. Les informations importantes sont ensuite récupérées, mises en forme équivalente adaptée à la comparaison, puis comparées à l'aide de fonctions bash.

Le nombre total de tests effectués et réussis est retourné à l'utilisateur pour analyser les résultats. Les tests affichent également des informations clé lorsqu'un test échoue, dans le but de débugger le programme principal.

Fonctionnalités implémentées

| Première phase .

ÉTAPE 1 .

Lecture de l'entête d'un fichier ELF – Fait
Affichage de l'entête d'un fichier ELF – Fait
Validation par test automatique – Faits

ÉTAPE 2 .

Lecture de la table des sections d'un fichier ELF – Fait
Affichage de la table des sections d'un fichier ELF – Fait
Validation par test automatique – Faits

ÉTAPE 3 .

Lecture du contenu d'une des sections d'un fichier ELF – Fait
Affichage du contenu d'une des sections d'un fichier ELF – Fait
Validation par test automatique – Faits

ÉTAPE 4 .

Lecture de la table des symboles d'un fichier ELF – Fait
Affichage de la table des symboles d'un fichier ELF – Fait
Validation par test automatique – Faits

ÉTAPE 5 .

Lecture de la table de réimplantation d'un fichier ELF – Fait
Affichage de la table de réimplantation d'un fichier ELF – Fait
Validation par test automatique – Faits

| Seconde phase .

ÉTAPE 6 .

Modification de l'organisation des sections d'un fichier ELF : commencé non terminé.
Tests : non implémentés.

ÉTAPE 7 A 11 .

Non implémentés.

Liste des bogues connus

À ce jour, aucun bogues connu.

Tests

Descriptions des différents tests effectués à chaque étape du programme :

Nous avons différents fichiers ELF (créés lors de la compilation) dans le dossier *Examples_loader* afin de tester toutes les étapes séparément (quand elles ne font pas appel à une étape précédente). Sachant que toutes les étapes dépendent au moins de la première, qui permet de remplir la variable header contenant les informations de l'en-tête (type Elf32_Ehdr de la documentation). Chaque test opère sur tous les fichiers *****.o et compte les tests reussis/échoués.

| Première phase .

ÉTAPE 1 .

Vérification de l'en-tête des fichiers ELF : *test_header.sh*
Utilisation de **diff -w** pour comparer la sortie standard de **readelf -h [file]** avec celle de **./read_elf -h [file]**

ÉTAPE 2 .

Vérification de la table de sections : *test_sect_head.sh*
Utilisation de **sed -n {x}p [file]** pour la lecture des lignes successives, puis pipe vers **tr -d ' '** et **cut -d' ' -f 6-** pour récupérer les sections importantes de la sortie. Ensuite, la comparaison une à une des sorties standard **readelf -S [file]** et **./read_elf -S [file]**

ÉTAPE 3 .

Vérification du contenu d'une section : *test_contenu_section.sh -x [file]*
Utilisation de **sed -n {x}p [file]** pour la lecture des lignes successives, et recherche des lignes commençant par "**0x**" qui sont significatives. Test des cas où la sortie standard ne contient aucune donnée (lorsque la section est vide). Ensuite utilisation de pipe vers **rev | cut -c 17- | rev** et **tr -d ' '** pour tronquer les 16 caractères de traduction hex-ascii de hexdump dans la sortie **readelf**. Ensuite toutes les lignes d'une section commençant par "**0x**" sont comparées.

ÉTAPE 4 .

Vérification de la table des symboles : *test_table_symbole.sh*
Utilisation simple de diff entre les sorties standard de **readelf -s [file]** et **./read_elf -s [file]** avec cat pour un formatage 1 mot/ligne. Le fichier **diffout** généré n'est jamais vide car **./read_elf** n'affiche pas le nom des sections 'usuelles' alors que **readelf** le fait ; le script ignore les lignes de **diffout** où Name commence par un point (ex .data)

ÉTAPE 5 .

Vérification de la table de réimplantation : *test_reloc.sh*
Utilisation simple de diff entre les sorties standard de **readelf -s [file]** et **./read_elf -s [file]** avec **cat** pour un formatage 1 mot/ligne. Là encore le script ignore les différences mineures d'affichage et ne prend en compte que les mots significatifs. Ajout d'un cas de test pour les fichiers sans relocations : on vérifie que les sorties **readelf -r [file]** et **./read_elf -r [file]** sont non-vides avant de procéder.

| Seconde phase .

ÉTAPE 6 à 11 .

Pas de tests car non-implémentés (exception pour l'étape 6 qui n'a pas de test car non-implémenté totalement)