

# Compte-rendu

[L3-INFO] PROG5

Réalisation d'un éditeur de liens — Phase de Fusion

Groupe n°6 :

CHARREL Vincent

CHOSSON Aurélien

DEPLANQUE Mathias

VAUDEY jolahn

VINSON Charly

YVON Alexis

<b>Compilation et Lancement</b>	<b>3</b>
<b>Descriptifs de la structure du code</b>	<b>4</b>
<b>Fonctionnalités</b>	<b>8</b>
Fonctionnalités Présentes	8
Fonctionnalités Manquantes	9
<b>Bogues connus et non résolus</b>	<b>10</b>
<b>Tests effectué</b>	<b>11</b>
<b>Journal de bord</b>	<b>12</b>

## I. Compilation et Lancement

Pour lancer le projet, il suffit simplement de suivre une suite d'étapes :

- 1 - Cloner le projet depuis GIT,
- 2 - Utiliser la commande 'make distclean' dans le répertoire 'elf\_linker 1.0',
- 3 - Utiliser la commande 'autoreconf' dans le répertoire 'elf\_linker 1.0',
- 4 - Utiliser la commande './configure' pour faire l'Automake du projet,
- 5 - Utiliser la commande 'make' afin de réaliser les exécutables du projet,
- 6 - Lancer l'exécutible correspondant à l'étape que l'on veut.

Liste des étapes et de leur exécutable :

ETAPES	Exécutable	arguments
1	readelf_header	<fichier elf>
2	readelf_section_list	<fichier elf>
3	readelf_section_content	<fichier elf>, "name" et le nom de la section OU "id" et l'id de la section
4	readelf_symbols_table	<fichier elf>
5	readelf_relocation	<fichier elf>
6	etape6	<fichier elf> <fichier elf>
7	etape7	<fichier elf> <fichier elf>

## II. Descriptifs de la structure du code

Voici la description de chaque fichier ainsi que ses principales fonctions.

### **Header** (header.c / header.h)

Ce fichier contient 2 fonctions : `read_header` qui lit l'entête d'un fichier `elf` placé en paramètre. Elle vérifie que le fichier est bien un fichier elf, vérifie l'endianness du fichier et compare à l'endianness de la machine, si les 2 sont différents alors on inverse l'endianness du fichier. La fonction lit le fichier elf au format 64 bits, si la variable à lire est une variable 32 bits on remplit avec des '0'. La fonction retourne un "`Elf64_Ehdr`" qui est une structure au format d'un entête de fichier elf.

La deuxième fonction : `print_header` affiche ce que la fonction précédente a retourné selon un format particulier, imitant `readelf -h` autant que possible.

### **Table des sections** (tablesection.c / tablesection.h)

Ce fichier contient 2 structures permettant de définir une liste de sections (`section_name` et `section_list`). De plus, il contient 2 fonctions : `read_tablesection` qui prend en entrée un fichier elf et un entête au format `Elf64_Ehdr` et retourne la liste des sections de ce fichier. Comme pour les fichiers précédents, on compare l'endianness du fichier avec celui de la machine et on l'inverse si ils sont différents.

La deuxième fonction : `print_sectionTable`, affiche les résultats de la fonction précédente en prenant en paramètre une liste de section.

### **Contenu des Sections** (readsection.c / readsection.h)

Ce fichier utilise les structures fournies par les deux fichiers précédents, et contient deux fonctions: `readsection_id` et `readsection_name`. Ces deux fonctions lisent et affichent le contenu d'une section spécifique du fichier ELF donné en argument, en utilisant la première on indique l'id de la section qui nous intéresse, tandis qu'en utilisant la seconde, on doit indiquer son nom. Ici,

l'endianness n'a pas vraiment d'importance, puisque l'on affiche le résultat byte par byte.

### Tables de symboles (symboltable.c / symboltable.h)

Ce fichier utilise les structures fournies par les étapes 1 et 2, et contient une structure `symbol_table_64` permettant de stocker la table de symbole, sa table de chaînes de caractères associée, et toutes les autres informations qui seront nécessaires pour réaliser la fusion (deux structures auxiliaires, `oldid` et `string_table` sont également décrites dans le .h). Le fichier contient également deux fonctions: `read_symbols_tables_64` qui lit la table de symbole du fichier ELF donné en argument et le stocke dans une structure, puis la renvoie, et `print_symbol_table_64` qui affiche la table de symbole donnée en argument sous forme de structure, à la façon de `readelf -S`.

### Tables des réallocations (realloc.c / realloc.h)

Ce fichier utilise les structures fournies par les étapes 1 et 2, et contient deux structures permettant de stocker respectivement une table de réallocation et les informations nécessaires à leur fusion, ou bien la liste des tables de réallocations d'un fichier ELF. Le fichier contient également quatres fonctions:

- `read_rela_tables_64` qui permet de remplir puis renvoyer une structure décrivant la table de réallocation dont l'index est passé en argument.
- `search_reloca_tables_64` qui remplit une structure de liste de tables de réallocations. La fonction détecte en parcourant une structure contenant la table des sections les différentes tables de réallocations, et utilise alors la fonction précédente `read_rela_tables_64` pour obtenir une structure décrivant la table en question, puis l'ajoute dans sa liste.
- `print_rela_64` qui affiche une table de réimplémentation, donnée en argument sous forme de structure, en imitant le résultat de `readelf -r`.
- `print_list_rela_64` qui affiche une liste de tables de réimplémentation, donnée en argument sous forme de structure, en utilisant la fonction précédente (`print_rela_64` ).

## Fusion des sections PROGBITS (progbits.c / progbits.h)

Ce fichier utilise les structures fournies par les étapes 1 et 2, et contient trois structures: Une première , Merge\_table\_progbits permet de retenir quelles sections du fichier 2 ont été fusionnées dans quelles sections du fichier 1. La seconde, Section\_progbits, permet de retenir le contenu d'une section de type PROGBITS. La troisième, Table\_sections, contient la représentation de la liste des sections de type PROGBITS d'un fichier ELF issus de la fusion des sections de deux fichiers ELF. Ce fichier contient également plusieurs fonctions, les plus importantes étant:

- affiche\_table\_section, qui affiche la table des sections PROGBITS issue d'une fusion.
- search\_progbits\_f2, qui remplit la structure Merge\_table\_progbits pour déterminer quelles sections doivent être concaténées.
- get\_merged\_progbits, qui réalise la fusion des sections PROGBITS des deux fichiers ELF reçus en arguments, et renvoie une structure Table\_sections contenant le résultat de la fusion.

## Fusion des tables de symboles (fusionsymbole.c / fusionsymbole.h)

Ce fichier utilise les structures issues des étapes 4 et 6, et contient 1 fonction, fusion\_symbol\_tables\_64, qui prends en entrée deux tables de symboles, et le résultat de la fusion de l'étape précédente (une structure Table\_sections ), et renvoie une structure de type symbol\_table\_64 contenant le résultat de la fusion des tables de symboles données en arguments.

De plus, il reste les fichiers des exécutables contenant le main pour chaque étape :

### **Etape 1 (readelf\_header.c)**

Ce programme prend en argument un fichier ELF, et affiche, à la façon de readelf -h, son entête à l'écran.

## Étape 2 (readelf\_section\_list.c)

Ce programme prend en argument un fichier ELF, et affiche, de façon similaire à readelf -S, la liste des sections et de leurs propriétés à l'écran.

## Étape 3 (readelf\_section\_content.c)

Ce programme prend en argument un fichier ELF, puis soit “name”, suivi d'un nom de section, soit “id” suivi d'un id de section. Ensuite, le programme affiche le contenu de la section demandée en hexadécimal et en ascii, de façon similaire à readelf -x.

## Étape 4 (readelf\_symbols\_table.c)

Ce programme prend en argument un fichier ELF, et affiche, de façon similaire à readelf -s, la table de symbole du fichier à l'écran, puisque nous ne sommes supposés n'en avoir qu'une.

## Étape 5 (readelf\_relocation.c)

Ce programme prend en argument un fichier ELF, et affiche ses tables de réimplémentation à l'écran. Pour chaque entrées, comme demandé dans le sujet, on affiche l'offset de réimplémentation, le type de réimplémentation (sous forme de nombre, puisque l'interprétation est laissée au compilateur) et l'index de l'entrée dans la table des symboles (ainsi que l'index de la table des symboles elle-même dans la table des sections en bonus).

## Étape 6 (etape6.c)

Ce programme prend en argument deux fichiers ELF, réalise la fusion de leurs sections de type PROGBITS de même nom et affiche à l'écran les sections de type PROGBITS qui apparaîtront dans le fichier ELF fusionné, ainsi que leur contenu.

## Étape 7 (etape7.c)

Ce programme prend en argument deux fichiers ELF, réalise la fusion de leurs sections de type PROGBITS de même nom, puis utilise le résultat pour réaliser la fusion de leurs tables de symboles, puis affiche les tables de symboles initiales suivies de la table de symboles résultant de la fusion à l'écran.

### III. Fonctionnalités

#### A. Fonctionnalités Présentes

1. **Affichage de l'entête** : permet d'afficher l'entête d'un fichier elf en vérifiant au préalable si c'est bien un fichier elf et son endianness.
2. **Affichage de la table des sections** : affiche la table des sections d'un fichier elf et les caractéristiques de chacune d'elle.
3. **Affichage du contenu d'une section** : affiche le contenu d'une section donnée en paramètre en l'identifiant par son nom ou par son id.
4. **Affichage de la table des symboles** : affiche la table des symboles d'un fichier elf et les caractéristiques de chacun d'eux.
5. **Affichage de la table de ré-implémentation** : affichage des tables de ré-implémentations d'un fichier elf et des détails relatifs à chaque entrées
6. **Fusion et re-numérotation des sections** : concatène les sections de type PROGBITS de deux fichiers elf et affiche le résultat.
7. **Fusion, re-numérotation et correction des symboles** : construction de la table des symbole en fusionnant les tables de symboles des deux fichiers en entrée, tout en prenant en compte la fusion effectuée à l'étape précédente. (Possède des bugs de mémoire pour le moment)

## **B. Fonctionnalités Manquantes**

- 1. Fusion et correction des tables de réimplantations**
- 2. Production d'un fichier résultat au format ELF**

#### IV. Bogues connus et non résolus

## V. Tests effectué

Afin de vérifier que les équivalents de readelf effectué soit correctes, ils ont été comparé avec les résultats du readelf ARM. Une fois les résultats des deux méthodes générés, il ont été comparé à la main étant donné que leurs affichages sont différents.

Pour les deux parties de la phase 2 réalisées, les tests n'ont pas eu le temps d'être entièrement réalisés, mais dans l'idée:

- Pour l'étape 6: on vérifie avec la table des sections (readelf -S ou équivalent) que toutes les sections progbits identiques présentes dans les deux fichiers sont bien fusionnées, et que les autres sont bien présentes, et on vérifie que le contenu des sections fusionnées correspond bien à la concaténation du contenu des deux sections (à l'aide d'un readelf -x ou équivalent)
- Pour l'étape 7: Le test est plus ou moins intégré au programme, puisqu'il affiche les tables de symboles des deux fichiers ELF puis celle résultant de la fusion. On peut donc vérifier que tous les symboles locaux sont présents, et que tous les symboles globaux sont présents une fois, et définis correctement s'ils le sont dans un des deux fichiers ELF en entrée (et bien sur que cela s'arrête si les deux fichiers ont un symbole global de même nom, main par exemple).

## VI. Journal de bord

Voici le journal du travail effectué par jour et par personne.

12/12/2019

Toute l'équipe : Lecture et prise de connaissance de l'ensemble du projet.  
Consultation des différentes documentations associées.

13/12/2019

Toute l'équipe : Réunion avec l'ensemble du groupe pour discuter des tâches à accomplir et de l'organisation du travail. Mise en place du GIT et présentation du client GIT “GIT KRAKEN”.

16/12/2019

Alexis : Essais de lecture d'un fichier ELF octet par octet ; corrections sur la lecture et l'affichage de l'entête d'un fichier ELF (étape 1) ; mise en place d'une nomenclature pour les commit sur GIT.

Aurélien : Travail sur l'étape 2.

Charly : Compréhension du sujet.

Jolahn : Réalisation d'une première version de la lecture et affichage du header d'un fichier ELF (étape 1) suivie de la réalisation d'une première version de la lecture et affichage des tables de sections (étape 2).

Mathias : Mise en place de GIT et explications de son fonctionnement (le maintien de git a été fait tout au long du projet). Répartitions des tâches. Prise de connaissance sur le projet (compréhension)

Vincent : Premier test sur les readelf (h et S) disponibles.

**17/12/2019**

Alexis : Corrections sur les fichiers concernant l'entête, la table des sections et celle des symboles (étapes 1 à 4).

Aurélien : Travail sur l'étape 2.

Charly : Compréhension du sujet.

Jolahn : Réalisation d'une première version de la lecture et affichage du contenu des sections d'un fichier ELF (étape 3), puis réalisation d'une première version de la lecture et affichage de la table des symboles (étape 4)

Mathias : Prise de connaissance de l'étape 1( ce qui a été fait) et aide sur l'optimisation. Prise de connaissance de l'étape 2 (ce qui a été fait) et aide sur l'optimisation.

Vincent : Début de travail sur l'étape 4 (table des symboles).

18/12/2019

Alexis : Harmonisation de la mise en forme, clarification et correction des fichiers concernant les étapes déjà traitées (1 à 4) ; création d'un Makefile temporaire en attendant de configurer l'Automake.

Aurélien : Malade

Charly : Compréhension du code effectué et modularisation de l'étape 1.

Jolahn : Réalisation d'une première version de la lecture et de l'affichage des tables de ré-implémentations (étape 5). Début du travail sur l'étape 6 (fusion des sections de type progbits).

Mathias : Aide diverse sur le projet (Optimisation, ...). Prise de connaissances du travail effectué (compréhension du code existant).

Vincent : Écriture des premiers scripts de tests (test.sh et exemples.sh).

19/12/2019

Alexis : Passage du Makefile temporaire à l'Automake initialement prévu ; difficultés à faire fonctionner celui-ci dues à plusieurs problèmes (caractères non reconnus, conflit Windows-Linux au niveau du format de certains fichiers, dépendances manquantes...).

Aurélien : Malade

Charly : Compréhension du code effectué et modularisation de l'étape 1.

Jolahn : Première version de l'étape 6 réalisée, dont l'exécutable fusionne les sections de type PROGBITS de deux fichiers ELF en entrée. Cette version n'utilise alors pas les résultats de la phase 1, puisque les programmes en questions ne font alors qu'afficher.

Mathias : Aide diverse sur le projet (Optimisation, ...). Prise de connaissances du travail effectué (compréhension du code existant).

Vincent : Première recherche pour faire fonctionner l'automake. Rattrapage d'erreur (fichier disparu) et premier renommage des fichiers, ainsi que modification du Makefile temporaire en conséquence.

20/12/2019

Alexis : Audit de code ; continuation des recherches pour résoudre les problèmes liés à l'Automake.

Aurélien : Audit de code.

Charly : Compréhension du code effectué et modularisation des étapes. Explication aux autres membres du groupe le sujet et les codes produits.

Jolahn : Audit de code le matin, suivi du début de travail sur l'étape 7 (fusion des tables de symboles).

Mathias : Audit de code.

Vincent : Audit de code.

06/01/2020

Alexis : Aide à la préparation du compte-rendu ; continuation des recherches pour résoudre les problèmes liés à l'Automake ; début de correction pour les étapes 5 et 6

Aurélien : Absent.

Charly : Modularisation des étapes 2 et 3.

Jolahn : Première version de l'étape 7 réalisée, qui permet de fusionner les tables de symboles de deux fichiers ELF en entrée, après que la fusion de l'étape 6 soit réalisée.

Mathias : Préparation du compte-rendu ; jalon sur l'avancement du projet.

Vincent : Amélioration et correction du Makefile temporaire

07/01/2020

Alexis : Automake opérationnel ; corrections ponctuelles sur des fichiers contenant du code mort et diverses erreurs.

Aurélien : Aide la rédaction du compte-rendu.

Charly : Prise de connaissance du code existant. Modularisation des étapes 2 et 3

Jolahn : Début du travail sur l'étape 8, améliorations des fonctions des étapes 4, 5 et 7 (modularisation).

Mathias : Rédaction du compte-rendu.

Vincent : Modification des scripts de test suite à l'automake opérationnel

08/01/2020

Alexis : Retour sur les étapes 1 (entête) et 2 (table des sections) pour ajouter et corriger des éléments (répétitions, commentaires...) ; mise à jour de l'Automake pour s'adapter aux étapes effectuées.

Aurélien : Aide à la rédaction du compte-rendu.

Charly : fin étape 3 essaie de l'étape 5 , modularisation de l'étape 6

Jolahn : Amélioration des étapes 4 et 5, et par extension de l'étape 7, Aide à l'amélioration de l'étape 2. Aide à la rédaction du compte-rendu.

Mathias : Rédaction du compte-rendu et préparation de l'oral. Aide pour l'optimisation du code.

Vincent : Continuation des tests, entre autre sur les readelf modularisés.

09/12/2020

Alexis : Dernières mises à jour du code et aide à la rédaction du compte-rendu ; préparation de l'oral.

Aurélien : Aide à la Rédaction du Compte rendu

Charly : test étape 6, préparation du journal de bord et création du premier couple de fichier pour tester étapes 6 et 7.

Jolahn: Améliorations des étapes 6 et 7, aide à la rédaction du compte-rendu, préparation de l'oral

Mathias : Rédaction du compte-rendu et préparation de l'oral.

Vincent : Production automatique des résultats de readelf\_section\_content (option x, étape 3) et readelf -x.