Raspail Clément
Lemay Louis
Barneaud Esteban
Savignon Thilen
Ciobanu Theodora
Ethève Alexis
Groupe 4

Projet PROG5: Compte Rendu

Fusion de fichiers ELF

Mode d'emploi:

Pour compiler et lancer les programmes utilitaires développés dans le cadre de ce projet, veuillez suivre les étapes suivantes:

Initialisation du projet

```
git clone https://github.com/AEtheve/Projet-PROG5
cd elf_linker-1.0

./configure CFLAGS='-Wall -Werror -Wno-unused-result -g'
automake --add-missing
make
```

Test

```
make check
python ./tests/test_fusion_section.py
```

Readelf

```
./readelf [ --help] [ --f <fichier> ] [ --h ] [ --S ] [ --s ] [
--x <section> ] [ --r ]
```

- `--help`: pour afficher les commandes disponibles
- `--fichier`: pour préciser le fichier à utiliser

- `--h`: pour afficher l'entête
- `--S`: pour afficher les sections
- `--s`: pour afficher la table des symboles
- `--x`: pour afficher la sections donnée en argument

Fusion

```
./fusion <fichier source 1> <fichier source2>
```

Descriptif de la structure du code développé:

Dans un premier temps, nous avons créé une structure globale contenue dans le module elf32 contenant des fonctions d'allocation, de réallocation et de libération de mémoire des différents types de notre structure. Nous avons introduit également les fonctions générales d'ouvertures et de fermeture de fichier dans util.c.

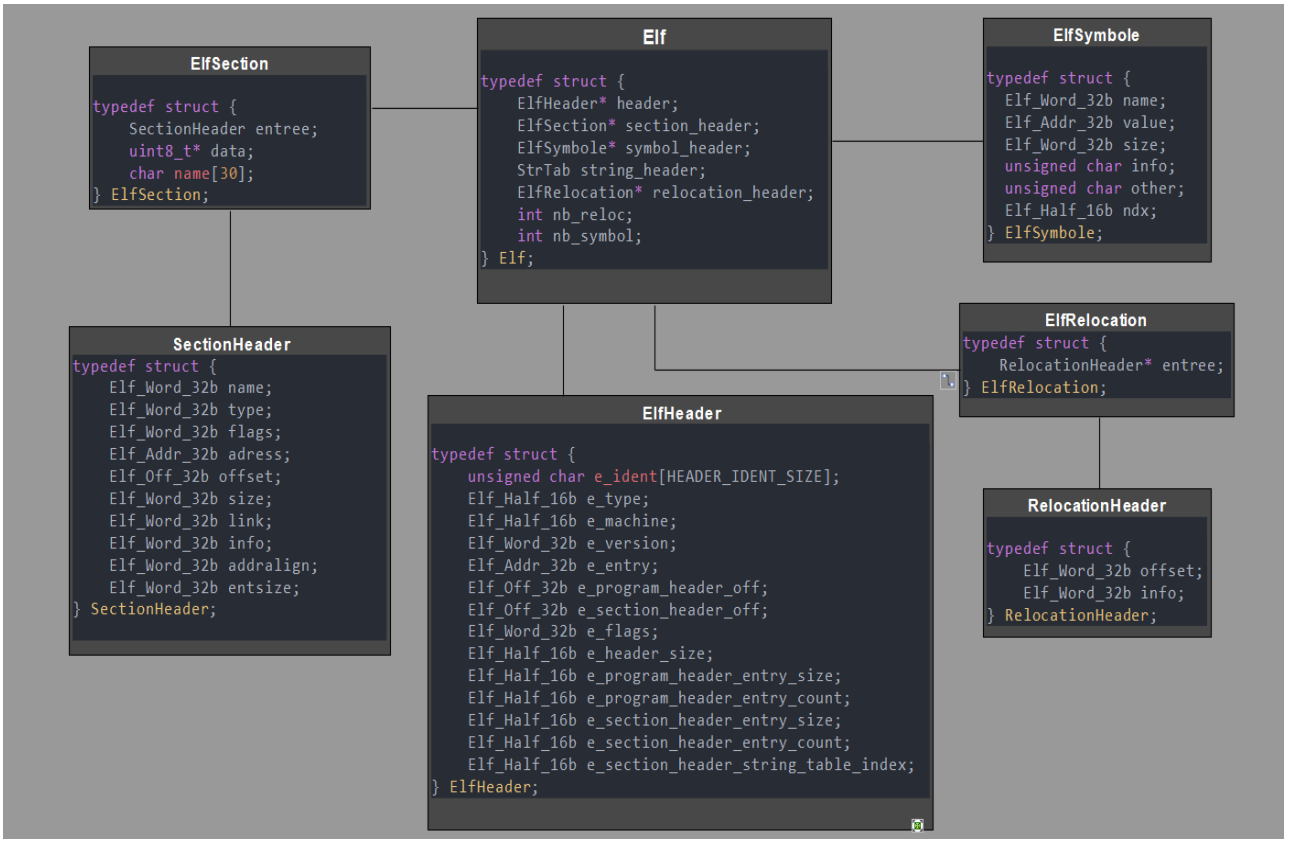

fig. 1: diagramme des différents types du module elf32

Par la suite, pour chaque partie du projet, nous avons implémenté des fichiers contenant les fonctions d'affichage et de création des structures:

- gestion_entete.c
- gestion_table_symboles.c
- gestion_contenu_section.c
- gestion_section.c
- gestion_table_relocation.c

Chacun des ces fichiers possède une fonction du même nom, qui ouvre le fichier, remplit notre structure, utilise la fonction d'affichage associé à la partie concernée

Ensuite, pour la fusion, nous avons utilisé nos structures précédemment établies.

- fusion.c
- fusion_relocation.c
- fusion_section.c
- fusion_table_symboles.c

Liste des fonctionnalités implémentées:

Voici une liste des fonctionnalités implémentées dans ce projet:

- affichage de l'entête d'un fichier ELF
- affichage des sections d'un fichier ELF
- affichage du contenu des sections d'un fichier ELF
- affichage de la table des symboles d'un fichier ELF
- affichage de la table de réimplantation d'un fichier ELF
- la fusion et la rémunération des sections
- la fusion et la correction des symboles
- la fusion et la correction des tables de réimplantations
- l'écriture d'un fichier résultat au format ELF

Nous avons implémenté toutes les fonctionnalités qui nous ont été demandées. Il nous aurait été possible de faire des fonctionnalités additionnelles, mais le temps ne nous a pas permis. Nous nous sommes donc limités aux fonctionnalités demandées afin de les réaliser correctement…

Liste des éventuels bogues connus mais non résolus:

Voici une liste des bogues connus dans ce projet:

- l'offset affiché et stocké dans la structure de fusion des tables de réimplantations est incorrect: nous n'avons pas trouvé de logique aux affichages de arm-none-eabi-readelf. Nous avons donc laissé ce que nous pensons être la méthode de calcul la plus logique.

Liste et description des tests effectués:

Voici une liste et une description des tests effectués sur ce projet:

- file1.c/file2.c: test fourni dans le squelette.

Nous avons testé notre code à la main avec 1 ou 2 autres fichiers mais nous ne les avons pas intégrés dans notre projet par manque de temps.

Tests automatiques

Nous avons mis en place des Tests automatiques en utilisant la librairie CuTest. Ces tests sont effectués pour la phase 1 (5 premières parties). Ils comparent les valeurs de retour de l'exécution des commandes "arm-none-eabi-readelf" avec le résultat de nos fonctions sur la sortie standard. Dans /tests/

- test_gestion_contenu_section.c
- test_gestion_gestion_entete.c
- test_gestion_section.c
- test_gestion_tr.c
- test_gestion_ts.c

Ces tests ont pour objectif de vérifier que les résultats de nos fonctions correspondent strictement au résultat attendu pour avoir de très bonnes bases pour la suite.
Pour les utiliser, il suffit d'utiliser la commande "make check".

Tests python

mode d'emploi test_fusion_section.py:
-> utiliser la commande **python3 tests/test_fusion_section.py** (on se place dans le dossier elf_linker-1.0)

description:
On crée une liste de listes pour stocker les sections qu'on obtient avec notre programme fusion_table_symboles.c et une autre pour stocker le résultat obtenu en utilisant la commande arm-none-eabi-readelf -S. On vérifie si les deux listes contiennent les mêmes valeurs.

Le test pour vérifier la bonne marche de la fusion des tables de réimplantation se fait avec le fichier relocV1.py.

Il suffit d'exécuter la commande suivante:

- **arm-none-eabi-readelf fichier_fusionné -r >arm.out** (pour créer l'affichage de fusion attendu)
- **clang fusion.c fusion_relocation.c fusion_section.c fusion_table_symboles.c gestion_*.c elf32.c util.c -o fusion_reloc** (pour compiler notre programme de fusion des tables de réimplantation)
- **./fusion_reloc fichier_1.o fichier_2.o** (pour créer notre fichier résultat)
- **arm-none-eabi-readelf out.o -r > out_fusion** (pour créer un fichier contenant l'affichage d'arm produit à partir de notre fichier résultat)
- **python3 reloc.py fichier1.o fichier2.o fichier_fusionné.o [-w]**
  On exécute ensuite notre test pour comparer ces 2 résultats et obtenir les différences.ARM L'option **'-w'** permet d'obtenir tous les détails des différences obtenuess' il