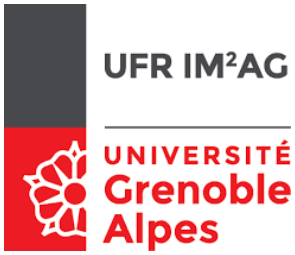

Rapport de projet – PROG5

Éditeur de liens

Thibault CHASSAGNOL

Sylvain JOUBERT

Oliver MINEAU

Vincent PENHOUËT

Antoine SAUZEAU

Alexis THACH

Responsable : Guillaume HUARD

3ème année de Licence Informatique

ANNÉE 2022-2023

TABLES DES MATIÈRES

I. Introduction

A. Contexte et motivation du projet

Notre projet consistait à réaliser la fusion de fichiers objet au format ELF, en utilisant un programme en langage machine ARM en entrée. L'objectif était de pouvoir lire la structure des fichiers objet, pour ensuite effectuer la fusion et produire un fichier objet unique. Il s'agissait de la partie initiale d'un éditeur de liens, permettant de lier des fichiers objet ensembles pour former un exécutable.

B. Objectifs du projet

Objectifs de programmation : Pour réaliser ce projet, nous avons dû développer un logiciel de taille moyenne.
Nous avons utilisé les compétences acquises pendant le semestre en programmation bas niveau et manipulation de la mémoire.

Objectifs de respect d'un cahier des charges : Nous avons dû apprendre à comprendre les documentations techniques volumineuses et à chercher les bonnes informations pour respecter les spécifications données.

Objectifs de génie logiciel : Nous avons dû apprendre à gérer le travail en groupe, en divisant le travail et en suivant l’évolution.

Objectifs de fonctionnalités : Produire un programme modulaire et efficace pour permettre la lecture et la réutilisation des données de fichier ELF.

II. Mode d'emploi

A. Prérequis techniques

Pour compiler et utiliser notre programme, il est nécessaire d'avoir un système d'exploitation compatible (Linux, base Unix) avec les commandes de compilation standard (gcc, make) et les options de compilation (CFLAGS). Il est également nécessaire pour les tests automatisés d'avoir linux en anglais (cf partie test).

B. Compilation

Pour compiler, il suffit de se rendre dans le répertoire où se trouve les fichiers source, et de lancer les commandes suivantes : CFLAGS=" ./configure, puis make. Cela compilera le programme, et il sera prêt à être utilisé.

C. Utilisation et interface utilisateur

Pour utiliser notre programme, vous avez le choix entre deux cas d'utilisation :

- Lecture d'un seul fichier ELF : Pour lancer cette option, utilisez la commande suivante :

  ./main <-l> <nom_fichier>

  Les options disponibles sont :
  - "-a" pour afficher toutes les parties du fichier
  - "-h" pour afficher l'en-tête d'un fichier ELF
  - "-S" pour afficher la table des sections d'un fichier ELF
  - "-s" pour afficher la table des symboles d'un fichier ELF
  - "-r" pour afficher les tables de réimplantation d'un fichier ELF pour machine ARM
  - "-x" pour afficher le contenu d'une des sections d'un fichier ELF, il est nécessaire d'ajouter un 4ème argument pour indiquer le numéro de la section à afficher :
    ./main <-l> <numero_section> <nom_fichier>

➢ Fusion de fichiers ELF : Pour lancer cette option, utilisez la commande suivante :

./main <-f> <nom_fichier_1> <nom_fichier_2> <nom_fichier_resultat>

Cela va fusionner les deux fichiers spécifiés et enregistrer le résultat dans le fichier spécifié en dernier argument (les trois fichiers doivent être de types objets “ELF”).

III. Structure du code

A. Description générale de la structure

La structure de notre code est basée sur une approche en deux points. La lecture, permettant de lire un fichier de type ELF et de stocker les informations nécessaires pouvant par la suite être ré-affichée de la même manière que readelf. La fusion, qui en fonction de deux fichiers, fusionne les informations nécessaires pour produire un résultat. La fonction principale "main" permet la gestion de l'exécution selon les arguments fournis par l'utilisateur.

B. Spécification des fichiers

Le fichier "lecture.c" contient les fonctions permettant de récupérer les informations du format ELF d'un fichier, de les stocker dans une structure et de les afficher selon les options données en argument. Cela inclut différentes fonctions, les principales étant la fonction d’initialisation de la structure ELF, de la lecture du fichier en mémoire à la structure ELF et de la fonction d’affichage global qui appelle les autres fonctions d'affichages (header, section).

Le fichier "fusion.c" contient la fonction principale “fusion()” qui étant donné trois fichiers (deux fichiers ELF à fusionner ouverts en lecture binaire, et un fichier de résultat ouvert en écriture binaire) récupère deux structures ELF et les fusionne pour former une structure ELF résultat, qui par la suite sera écrite dans le fichier résultat correspondant. La fonctions "fusion_sections_simple concat()”

permettant de concaténer les sections lorsque cela est nécessaire et la fonction “fusion_symbol_tables()” fusionnent les tables des symboles à l’aide de la fonction.

IV. Fonctionnalités implémentées

A. Description détaillée des fonctionnalités

Fonctionnalitées implémentées :

➢ Lecture du contenu d'un fichier au format ELF

- Lecture et affichage de l’en-tête
- Lecture et affichage de la table des sections
- Lecture et affichage du contenu d’une section
- Lecture et affichage de la table des symboles
- Lecture et affichage des tables de réimplantation

➢ Fusion de deux fichiers ELF

- Fusion et numérotation des sections
- Fusion, numérotation et correction des symboles

Fonctionnalitées manquantes :

- Fusion et correction des tables de réimplantations
- Production d’un fichier résultat au format ELF

B. Illustration des résultats obtenus

```
Affichage du header :

ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          592 (bytes into file)
  Flags:                             0x5000000, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         11
  Section header string table index: 10
```

Affichage de la table des sections :

```
There are 11 section headers, starting at offset 0x250:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000040 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 0001dc 000020 08   I  8   1  4
  [ 3] .data             PROGBITS        00000000 000074 000000 00  WA  0   0  1
  [ 4] .bss              NOBITS          00000000 000074 000000 00  WA  0   0  1
  [ 5] .rodata           PROGBITS        00000000 000074 00000a 00   A  0   0  4
  [ 6] .comment          PROGBITS        00000000 00007e 000034 01  MS  0   0  1
  [ 7] .ARM.attributes   ARM_ATTRIBUTES  00000000 0000b2 00002a 00      0   0  1
  [ 8] .symtab           SYMTAB          00000000 0000dc 0000e0 10      9  11  4
  [ 9] .strtab           STRTAB          00000000 0001bc 00001f 00      0   0  1
  [10] .shstrtab         STRTAB          00000000 0001fc 000051 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), y (purecode), p (processor specific)
```

Affichage du contenu d'une section :

```
Hex dump of section '.text':
 NOTE: This section has relocations against it, but these have NOT been applied to this dump.
  0x00000000 e92d4800 e28db004 e24dd010 e51b100c .-H......M......
  0x00000010 e51b0008 ebfffffe e50b0010 e51b1010 ................
  0x00000020 e59f0014 ebfffffe e3a03000 e1a00003 ..........0.....
  0x00000030 e24bd004 e8bd4800 e12fff1e 00000000 .K....H../......
```

Affichage de la table des symboles :

```
Symbol table '.symtab' contains 14 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS test1.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 00000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 00000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 00000000     0 SECTION LOCAL  DEFAULT    5 .rodata
     6: 00000000     0 NOTYPE  LOCAL  DEFAULT    5 $d
     7: 00000000     0 NOTYPE  LOCAL  DEFAULT    1 $a
     8: 0000003c     0 NOTYPE  LOCAL  DEFAULT    1 $d
     9: 00000000     0 SECTION LOCAL  DEFAULT    6 .comment
    10: 00000000     0 SECTION LOCAL  DEFAULT    7 .ARM.attributes
    11: 00000000    64 FUNC    GLOBAL DEFAULT    1 main
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND max
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

Affichage des tables de reimplantation :

```
Relocation section '.rel.text' at offset 0x1dc contains 4 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000014  00000c1c R_ARM_CALL        00000000   max
00000024  00000d1c R_ARM_CALL        00000000   printf
00000038  00000028 R_ARM_V4BX
0000003c  00000502 R_ARM_ABS32       00000000   .rodata
```

V. Tests et validation

A. Description des tests effectués

Nous utilisons un script nommé "test.sh" qui permet d'automatiser les tests pour la lecture de fichier, pour cela nous comparons les sorties de notre programme aux sorties de la commande readelf. Pour exécuter ce script il suffit de se placer dans le dossier courant du projet et de saisir dans un terminal Bash, par exemple, l'une des commandes ci-dessous en fonction du résultat souhaité.

- ➢ Tester tous les fichiers.o du dossier : Pour lancer cette option, utilisez la commande suivante :

  ./test.sh <dossier>

- ➢ Tester le fichier.o : Pour lancer cette option, utilisez la commande suivante :

  ./test.sh <fichier.o>

- ➢ Tester le fichier.o avec l'option debug : Pour lancer cette option, utilisez la commande suivante :

  ./test.sh <-d> <fichier.o>

Les sorties de notre programme sont en anglais ainsi, si l'on souhaite les comparer avec les sorties des commandes readelf elles doivent également être en anglais et donc linux doit être configuré en anglais.

Pour chaque étape, si les deux sorties sont les mêmes, le test est considéré comme réussi et l'étape est passée.

En cas de différences dans les sorties, une ligne rouge est affichée, représentant la sortie de notre programme. Sous chaque ligne rouge, la ligne attendue est affichée en vert. Le nombre d'erreurs trouvées dans cette partie est alors indiqué.

Une option de debug a été implémentée permettant d'afficher toutes les lignes qui sont présentes dans les deux résultats.

```
0x00000000 e92d4800 e28db004 ebfffffe e3a03000 .-H...........0.
0x00000010 e1a00003 e24bd004 e8bd4800 e12fff1e .....K....H../..

Hex dump of section '.rel.text':
  0x00000000 08000000 1c090000 1c000000 28000000 ............(...->3
  0x00000000 00000008 0000091c 0000001c 00000028 ...............(->3
Section '.data' has no data to dump.
Section '.bss' has no data to dump.

Hex dump of section '.comment':
```

B. Résultats des tests

```
pht20100@Ubuntu-20100:~/App/GitHub/Projet_Prog$ ./test.sh ./tests/test1BON.o
(Etape 1) : Test du header de test1BON.o :
Test header : test1BON.o REUSSI.
(Etape 2) : Test du section header de test1BON.o :
Test section header : test1BON.o REUSSI.
(Etape 3) : Test du contenu de section de test1BON.o :
Test content sec : test1BON.o REUSSI.
(Etape 4) : Test de la table de symbole de test1BON.o :
Test symb tab : test1BON.o REUSSI.
(Etape 5) : Test de la section de relocation de test1BON.o :
Test reloc : test1BON.o REUSSI.
```

Lors de la création d'un test on ajoute à son nom "BON" ou "MAUV" (mauvais) afin d'indiquer si le test doit renvoyer une erreur ou non.

Nom de fichier	Type d'erreur	Résultat du test
bon	pas d'erreur	bon
bon	erreurs de fichier elf	erreur
bon	autres erreurs	erreur
mauvais	pas d'erreur	erreur
mauvais	erreurs de fichier elf	bon
mauvais	autres erreurs	erreur

On vérifie donc chacun de nos tests en comparant leurs sorties avec la sortie voulue avec la commande:

./mainTest.sh <dossier>

```
Test debugBON.o : BON
Test file1BON.o : BON
Test file2BON.o : BON
Test ResultFile1-2BON.o : BON
Test ResultTest1-2BON.o : BON
Test test1BON.o : BON
Test test2BON.o : BON
Test test3MAUV.o : BON
Test test4BON.o : BON
Test test5BON.o : BON
Test test6MAUV : BON
Test test7MAUV.o : BON
Test test8MAUV.o : BON
13/13
```

VI. Conclusion

En conclusion, notre projet consistait à développer un logiciel de fusion de fichiers objet au format ELF, permettant de lier des fichiers objet ensemble pour former un exécutable. Nous avons travaillé en équipe pour réaliser cette tâche, en utilisant les compétences en programmation bas niveau et en utilisation de la documentation technique.

Malgré des difficultés rencontrées lors de la compréhension de certaines étapes du projet, nous avons réussi à compléter la plupart des étapes et fonctionnalités demandées. Nous avons effectué des tests automatisés pour valider les résultats obtenus et avons identifié des améliorations à apporter pour garantir la qualité du produit final.

Dans l'ensemble, nous sommes fiers de ce que nous avons accompli au cours de ce projet et nous sommes convaincus que les compétences acquises lors de ce projet nous seront très utiles dans notre futur professionnel.

VII. Références

Site du projet, https://prog5.gricad-pages.univ-grenoble-alpes.fr/Projet/

elf.pdf, https://prog5.gricad-pages.univ-grenoble-alpes.fr/Projet/elf.pdf

Arm7.pdf, https://prog5.grcocicad-pages.univ-grenoble-alpes.fr/Projet/arm7.pdf