Compte rendu

Réalisation d'un éditeur de liens - Réimplantation

Groupe 6 :

Mohamed BELHOUCHET

Hugo FETRE

Dorsaf KHEBTANI

Timothee ROGNON

Remi SAMSON

Mathis VILLARD

Bouchra ZAGHDAR

Table des matières

1. Descriptif de la structure du code

1.1. Liste des fonctionnalités implémentées

Pour la première phase du projet, voici une liste des fonctionnalités implémentées dans chaque fichier et leur rôle :

- *elf_header.h/c* fichier permettant la lecture et l'affichage de l'en-tête d'un fichier au format ELF.

Structure définie :
Pour mémoriser le nom des headers des sections.

```
typedef struct {
        Elf32_Shdr S;
        char *name;
} Elf32_ShdrNames;
```

Fonctions :
checkMagic( ) : contrôle de la valeur du magic (valeur fixée).
read_header( ) : lecture de l'en-tête d'un fichier ELF.
print_header( ) : l'affichage de l'en-tête d'un fichier ELF.
print_elf_header_complete( ) : l'affichage de l'en-tête ainsi que tous ses éléments.

- *elf_sections.h/c* fichier permettant la lecture et l'affichage de la table des sections.

Fonctions :
read_sections( ) : lecture de la table des sections.
print_sections_header( ) : l'affichage de la table des sections.
checkSectionTableIndex0( ) : vérifie que la section de l'indice 0 a les bonnes valeurs.
debugPrintSectionTableIndex( ) : fonction de debug, affiche les valeurs d'une section donnée en argument.
createSectionTable( ) : création de la table des sections.
freeSectionTable( ) : libération de la table des sections.
show_Name( ) : affichage du nom de la section.

- *elf_hedump.h/c*

Fonctions :
print_sections_hexdump() : l'affichage du contenu d'une section en hexadécimal.

- *elf_symtab.h/c* fichier permettant la lecture et l'affichage de la table des symboles.

Fonctions :

read_symtab( ) : lecture de la table des symboles.
print_symtab( ) : l'affichage de la table des symboles.
checkSymbolTableIndex( ) : vérifie que l'indice 0 de la table des symboles a les bonnes valeurs.
createSymNames( ) : création de la table des symboles.
freeSymNames( ) : libération de la table des symboles.

- *elf_reltab.h/c* fichier permettant la lecture et l'affichage de la table des réimplantations.

Fonctions :

read_reloc( ) : lecture de la table des réimplantations.
print_reloc( ) : l'affichage de la table des réimplantations.
free_reloc( ) : libérer le tableau alloué.

Pour ce qui est de la deuxième phase, nous avons les fonctionnalités suivantes :

- *elf_reimplantation.h/c* fichier permettant de créer un fichier .o à partir d'un autre .o en renumérotant les sections(ce qui modifie aussi la section symtab avec les adresses données en paramètre aux bons endroits) tout en appliquant la reimplantation de type ABS32/ABS16/ABS8 CALL, JUMP24 sur les bonnes sections.

Fonctions :

changeHeader( ) : crée le header du nouveau .o, le même sans les sections SHT_REL.
changeSections( ) : crée la table des sections du nouveau .o, le même sans les sections SHT_REL.
changeSymTable( ) : crée la table des symboles du nouveau .o, change les valeurs d'offset à valeur absolue.
writeFileHeader( ) : écris le nouveau header dans le 1er argument.
writeFileSectionHeader( ) : écris la nouvelle table des sections dans le 1er argument.
writeFileSection( ) : écris les anciennes sections au bons endroits dans le 1er argument.
writeFileSymTable( ) : écris la nouvelle table des symboles dans le 1er argument.
writeReimp( ) : effectue les modifications indiquée dans la table de réimplantation dans le 1er argument pour R_ARM_ABS8, R_ARM_ABS16 et R_ARM_ABS32.

1.2. Liste des fonctionnalités manquantes

Etant donné que nous avions rencontré des difficultés au niveau de la phase 2 partie 9, nous avons mis plus de temps que prévu pour développer le reste des parties. Par conséquent, certaines fonctionnalités n'ont pas pu être implémentées à temps. Ces fonctionnalités concernent l'interfaçage avec le simulateur ARM et la production d'un fichier exécutable non relogeable.

2. Mode d'emploi du projet

Pour compiler notre projet (soit pour la première phase ou la deuxième) on utilise les commandes suivantes : ./configure puis make.

Pour exécuter la première phase on utilise la commande :
./mainPrint (-x | -h | -r | -S | -s) /Examples_loader/example1.o (pour lancer sur l'exemple1, pour les autres on remplace avec le nom du fichier dans la commande)
--help -H

Pour exécuter la deuxième phase on utilise la commande :
./mainReloc ./"nomdufichierouappliquélesReloc" ./"nomdufichierResultat" .nomdesection1 addresse1 .nomdesection2 addresse2
Exemple :
./mainReloc ./Examples_loader/example3.o ./Examples_loader/ztest.o .text 32 .data 10240

3. Liste des bogues rencontrés et non résolus

Nous avons rencontré un bug au niveau de la partie 9 de la deuxième phase qui concerne les réimplantations de type R_ARM_JUMP24 et R_ARM8_CALL ; on écrit à la bonne adresse mais pas la bonne valeur.

4. Liste des tests effectués

Nos tests sont divisés en 2 parties :

Partie 1.1 : test.sh
Comparer les résultats de notre programme avec ceux de la commande *arm-none-eabi-readelf* pour vérifier que le traitement des données est correct. Pour cela, nous avons fait un script bash qui exécute séparément le programme et la commande *arm-none-eabi-readelf*, stocke les résultats de chaque exécution dans 2 fichiers textes puis les compare en ignorant les formalités de mise en forme (espaces, retours à la ligne, commentaires, upper/lower case, etc.). Le script affiche dans le terminal si le test est validé ou non. La différence entre les deux résultats, s'il y en a une, est affichée dans un fichier texte nommé *log.txt*.
Pour lancer le test: bash test.sh -opt fichier.o ou bash test.sh -x section fichier.o

Attention: le script est sensible aux différences d'orthographe. Des tests peuvent être invalides alors que la différence se situe uniquement au niveau des mots choisis pour l'affichage des données. En cas de doute, consulter le fichier log.txt.

Partie 1.2 :
Nous avons fait des fichiers .o avec des modifications pour voir le comportement du programme lorsqu'on lui passe un fichier qui n'est pas au format ELF ou qui contient une faute.

Partie 2 : testreloc.sh
Nous comparons 2 fichiers, l'un avant relocation et l'autre après. Nous n'avons pas eu le temps de bien rédiger cette partie des tests donc nous affichons simplement dans la console s'il y a des différences entre les 2 fichiers. Les différences peuvent être consultées dans les fichiers situés dans le dossier *test_logs* (à créer au préalable).
Pour lancer le test: bash testreloc.sh fichier1.o fichier2.o nomSection

5. Journal décrivant la progression du travail

Stratégie adoptée :
Etant donné que le projet est découpé en deux parties (phase 1 : lecture et affichage d'un fichier ELF, phase 2 : modifier le contenu du fichier ELF afin d'effectuer l'implantation), nous avons attribué la 1ère semaine du projet à la première phase et la 2ème semaine pour la deuxième phase.
Nous avons constaté que le développement d'une partie dépend de celle qui la précède, ce qui rend la répartition des tâches assez pénalisante. Nous avons donc décidé de travailler tous ensemble sur chaque partie. Des sessions de visioconférence via Discord et de liveshare sur Visual Studio Code nous ont été d'un très grand aide. Cette méthode s'est avérée très productive.

Phase 1 :
03/01/2022 : Prise en main de la documentation + réalisation de la 1er étape (elf_header et le main).
04/01/2022 : Finalisation de l'étape 1 (rédaction propre) + début de l'étape 2 et 3 (elf_sections).
05/01/2022 : Correction des erreurs et optimisation + changement de la structure du projet + démarrage de l'étape 4 (elf_symtab).
06/01/2022 : Améliorations du main et des premiers fichiers + réalisation des tests + audit + début de l'étape 5(elf_reltab).
07/01/2022 : Etape 5 achevée + correction des bugs + finition de la première phase.

Phase 2 :

08/01/2022 : Rédaction du compte rendu + tests automatiques de la phase 1.

09/01/2022 : Début de l'étape 6(renumérotation des sections) + l'étape 7(correction des symboles).

10/01/2022 : Etapes 6 et 7 achevées + début de l'étape 8(Reimplantation de type R_ARM_ABS*) + tests de la phase 2.

11/01/2022 : Finalisation de l'étape 8 + début de l'étape 9 (Reimplantation de type R_ARM_JUMP24 et R_ARM_CALL) + tests.

12/01/2022 : Correction et finition des étapes précédentes + Finalisation du compte rendu.