“ PROG5 - Projet Éditeur de Liens ”

Compte rendu

MILANOV Boyan - LAWSON Thibault - BOUZIGUES Alixia - ABDULLAH HASIM Mohd Thaqif - DIDIER Clément

1 Description du travail

1.1 Mode d'emploi

Afin d'exécuter le code fourni, il faut :

1. inclure le code source fourni par les enseignants ainsi que le simulateur ARM dans le dossier src/Programme_Editeur_Liens
2. lancer le programme launcher.sh situé à la racine du dossier. Celui-ci configure l'environnement et compile certains exécutables.
3. lancer le programme launcher.sh située dans le dossier src/Programme_Editeur_Liens, qui compile l'ensemble du projet, et lance les simulations sur l'ensemble des exemples contenus dans le sous-repertoire Example_loader. Si un nom de fichier objet est donné en argument, la simulation n'est lancée que pour ce fichier.

Si on souhaite compiler sans exécuter l'éditeur de liens, il suffit (après configuration de l'environnement de compilation croisée) d'exécuter un 'make' dans le dossier src/Programme _Editeur_Liens.

1.2 Structure du code développé

Le programme que nous avons développé pour effectuer l'édition de liens s'organise en différents modules :

1. API : il s'agit du module de base du programme. Il contient les définitions des structures de données utilisées, les fonctionnalités de lecture/écriture de fichiers au format ELF, des constructeurs et des accesseurs pour les structures de données définies, ainsi que les fonctions nécessaires pour libérer la mémoire à la fin de l'exécution. En particulier il permet de construire une représentation adaptée du ELF Header et des différentes sections (header + contenu) pour permettre ensuite de travailler dessus.
2. renumeration_section : ce module implémente l'étape 6 du sujet fourni, à savoir la suppression dans le fichier exécutable des sections servant à décrire les réimplantations à effectuer. La fonction à appeler est "enlever_relocation(...)". Elle fait appel à d'autres fonctions qui ne sont utilisées qu'à l'intérieur du module.
3. correction_symboles : ce module implémente l'étape 7 du sujet. D'une part, il a pour but de calculer les valeurs définitives des symboles en tenant compte des adresses de

chargement des différentes sections, d'autre part il modifie les indexes associés à chaque symbole selon la renumérotation des sections effectuée au préalable.

4. réimplantation : un module qui se charge d'effectuer les 3 types de réimplantations R_ARM_ABS*, R_ARM_JUMP24 et R_ARM_CALL. La procédure principale est simplement reimplantation() et elle utilise des sous-procédures distinctes pour chaque réimplantation.

5. Simulation : Le programme principal qui, grâce aux modules développés, lis un fichier objet ELF, et effectue les transformations nécessaires pour produire le fichier exécutable correspondant. Après production du fichier résultat, le code généré est simulé pas à pas via le simulateur arm.

1.3 Fonctionnalités implémentées

Les fonctionnalités de la partie 1 (lecture d'informations dans un fichier objet ELF) que nous avons implémentées sont les suivantes :

- Affichage de l'en-tête ELF
- Affichage de la table des sections
- Affichage du contenu d'une section
- Affichage de la table des symboles
- Affichage des tables de réimplantation

En ce qui concerne la partie 2, nous avons effectué :

- Renumérotation des sections
- Correction des symboles
- Réimplantations (les 3 types)
- Exécution de code dans le simulateur ARM

L'étape que nous n'avons pas eu le temps d'implémenter :

- Exécution du fichier résultat grâce à l'outil arm-eabi-run : nous n'avons pas construit la table des headers de programme, qui contiennent les informations sur les segments mémoire à allouer et les sections qu'elles contiennent.

1.4 Tests effectués

Le projet dispose de deux séries de tests respectivement disponibles dans les dossiers racine du projet et "src/Programme_Editeur_Liens".

La première est utilisée pour vérifier la correction des actions de la phase 1 du projet, c'est à dire la lecture des informations dans un fichier au format ELF. Un programme de test est disponible, il lit un fichier ELF et affiche toutes les informations relatives au header, à la table des sections, aux sections de réimplantations, et à la table des symboles.

La seconde batterie de tests permet de vérifier que les modifications du fichier ELF en vue de créer un exécutable sont correctes. Des exemples au format objet sont lus, l'édition de liens est effectuée, et le code obtenu est exécuté dans le simulateur ARM. Nous avons implémenté des exemples simples, dont la correction de l'exécution peut facilement être vérifiée, mais qui couvrent l'ensemble des cas possibles pour les réimplantations (R_ARM_ABS*, R_ARM_CALL, R_ARM_JUMP24).

Les tests sont lancés automatiquement par des scripts shell, qui configurent l'environnement, effectuent la compilation des différents fichiers, et lancent les exécutables nécessaires.

2 Journal

- Lundi 4 janvier 2016 :
  Prise en main et compréhension du sujet
  Distribution des tâches de la phase 1 :
  - Etape 1 (Affichage de l'en-tête) →Alixia et Thaqif
  - Etape 2 (Affichage de la table des sections) →Clément
  - Etape 3 (Affichage du contenu d'une section) →Thibault
  - Etape 4 (Affichage de la table des symboles) →Boyan
  - Etape 5 (Affichage des tables de réimplantation) →Alixia et Thaqif

  Début du projet
- Mardi 5 janvier 2016 :
  Réalisation des différentes étapes, test, correction des bugs.
- Mercredi 6 janvier 2016 :
  - Finalisation de la phase 1

  - Bilan du travail effectué jusqu'alors : nous avons fait le point sur le travail de chacun. Après avoir mis en commun nos différentes fonctions, nous nous sommes concertés pour trouver une approche adéquate des problèmes de la phase 2. Nous nous sommes alors répartis le travail à faire sur les étapes de la phase 2. Et nous avons convenu de rassembler nos travaux en une API commune qui aura servi de base commune au code développé a posteriori.
  - Création de l'API
  - Répartition des tâches pour les étapes 6 et 7 de la phase 2 (renumérotation des sections et correction de la table des symboles)

- Jeudi 7 janvier 2016 :
  Réalisation des étapes 6 et 7
  Mise à jour de l'API pour ajouter des fonctionnalités (affichage, écriture, manipulation de structures, ...)

- Vendredi 8 janvier 2016 :
  Test du code développé la veille
  Préparation de l'étape 8 : compréhension du principe des réimplantations
  Correction de bugs dans l'API

- Lundi 11 janvier 2016 :
  Audit de code a 10h30
  Répartition des tâches pour les jours suivants
  Modification du code suite aux remarques des professeurs : gestion du boutisme selon la machine cible, correction des procédures pour travailler sur des variables locales et non globales.

- Mardi 12 janvier 2016 :
  Fin de l'implémentation des étapes 8 et 9
  Finalisation des différentes tâches et test sur les fonctions terminées
  Compréhension du fonctionnement du simulateur ARM
  Tri et classement des sources du projet (dans le github)

- Mercredi 13 janvier 2016 :
  Correction de certains bugs inopinés
  Réalisation de l'étape 10 : simulation de notre programme sur les exemples fournis. Les résultats sont concluants.

- Jeudi 14 janvier 2016 :
  Création de différents tests pour l'éditeur de liens
  Écriture du compte-rendu