Guide d'exécution de binaires ARM avec QEMU sur Linux

Merci à Julien ZGUR (étudiant en L2 MIN 2024-2025) pour cette proposition de documentation.

Mise en garde sur l'utilisation de votre machine personnelle

Attention, **cette documentation s'adresse à celles et ceux qui sont suffisamment à l'aise avec leur ordinateur.** Aucun support ne sera apporté, la méthode à privilégier restant le travail à distance via `ssh` sur `turing`.

De plus, l'exécution peut un peu varier entre l'énoncé du TD et votre exécution avec `qemu` (mais rien de grave en principe).

Table des matières

Introduction

Ce guide explique comment exécuter des binaires compilés pour l'architecture ARM sur des machines Linux en utilisant QEMU un émulateur de processeur (ici en émulant un proc arm) et un compilateur adapté. Il vous permettra aussi de pouvoir debuggé vos programmes en assembleur.

Prérequis

Ce guide est dédié seulement à quelques (les plus utilisés) gestionnaire de paquets. Donc il vous **faudra** avoir une machine avec un gestionnaire de paquets qui se trouve dans cette liste : apt (Debian/Ubuntu/Mint), pacman (Arch Linux), zypper (openSUSE).

Installation de QEMU et du Compilateur ARM

Packet Manager	Émulateur Qemu	Compilateur ARM	Debuggeur (optionel)
APT	qemu-user	gcc-arm-linux-gnueabi	gdb-multiarch
Pacman	qemu-user	arm-linux-gnueabi-gcc	
Zypper	qemu-user	gcc-arm-linux-gnueabi	

APT

Avant de commencer, quelques explications :

J'ai choisis gcc-arm-linux-gnueabi à la place de gcc-arm-non-eabi, car je compte compiler et exécuter mes programmes sur une machine avec un noyau (systeme d'exploitation) linux, ce qui veut dire que mon processeur a accès à des "syscall" (read,write, etc). De plus la différence de bibliothèques C est un élément important, gcc-arm-linux-gnueabi à comme bibliothèque glibc contrairement à gcc-arm-non-eabi qui à une bibliothèque minimale. Pour finir la version linux, comme son nom l'indique, cible des processeurs ARM avec Linux, alors que la version none cible des microcontroleurs ce qui nous intérèsse pas vraiment.

Du côté de l'émulation, qemu-user permet d'émuler dans notre cas: le processeur (architecture ARM) et les instructions (qu'il). Alors que par exemple qemu-system lui émule toute une machine de A à Z, ram, disque, le bootloader, les périphériques, etc. Ce qui veut dire que utiliser qemu-system il nous faudrait une image du noyau qui nous intéresse, tout cela est bien trop compliqué et non adapté à l'utilisation que l'on veut faire.

Ceci étant dit place à l'installation des outils qui nous seront utiles au long de se semestre. **AVANT** toute chose si cela n'a pas été fait récemment, il faut mettre à jour la liste de paquets de apt (ce qui évitera d'avoir des erreurs lors de l'installation des futurs paquets requis) avec:

```
sudo apt update
```

Maintenant pour installer le cross-compileur : (nom de **l'executable** arm-linux-gnueabi-gcc)

```
sudo apt install gcc-arm-linux-gnueabi
```

Et pour l'émulateur qemu il vous faut deux paquets : *qemu-user* et *qemu-user-binfmt* (Dépendant de *qemu-user*)

```
sudo apt install qemu-user
```

Pacman

AVANT toute chose si cela n'a pas été fait récemment, il faut mettre à jour la liste de paquets de pacman (ce qui évitera d'avoir des erreurs lors de l'installation des futurs paquets requis) avec:

```
sudo pacman -Sy
```

Maintenant pour installer le compilateur :

```
sudo pacman -S arm-linux-gnueabi-gcc
```

Et pour l'émulateur qemu il vous faut deux paquets : ***qemu-user*** et ***qemu-user-binfmt*** (Dépendant de *qemu-user*).

```
sudo pacman -S qemu-user
```

Zypper

AVANT toute chose si cela n'a pas été fait récemment, il faut mettre à jour la liste de paquets de zypper (ce qui évitera d'avoir des erreurs lors de l'installation des futurs paquets requis) avec:

```
sudo zypper refresh
```

Maintenant pour installer le compilateur :

```
sudo zypper install gcc-arm-linux-gnueabi
```

Et pour l'émulateur qemu il vous faut deux paquets : ***qemu-user*** et ***qemu-user-binfmt*** (Dépendant de *qemu-user*).

```
sudo zypper install qemu-user
```

Utilisation de base

Pour la suite il se peut que la commande pour compiler ne commence pas de la même manière sur votre machine (et oui comme les paquets n'ont pas le même nom d'un gestionnaire à l'autre). Les futures commandes exposés jusqu'à la fin de ce guide seront pour des paquets provenant d'**APT**. (C'est a dire que de votre côté il vous faudra remplacer *arm-linux-gnueabi-gcc* par le nom de l'executabe qui a été installé avec le paquet, si vous êtes passé par un **autre** gestionnaire de paquets qu'APT).

La façon de compiler est similaire à celle pour compiler des fichiers ".c" avec gcc.

```
arm-linux-gnueabi-gcc <file0.s> ... <fileN.s> -o <output_nameFile>
```

Exécution du binaire que vous aurez nommé comme vous vous voulez (à la place de) produit par la commande ci-dessus. Attention l'option -L permet de spécifier le prefix du bon interpreteur pour les fichier elf :

`qemu-arm -L /usr/arm-linux-gnueabi/ ./<output_nameFile> <arg0> ... <argN>`

Et voila !!! Vous savez maintenant comment exécuter du code arm sur votre machine personnel.

Si une de ses deux commandes produit une erreur non liée au fonctionnement de votre programme (Command 'arm-linux-gnueabi-gcc' not found et/ou qemu-arm: Could not open '/lib/ld-linux.so.3': No such file or directory") aller voir dans la rubrique Ressources supplémentaires

Débug

Pour pouvoir debuguer un tel programme, il vous faudra "lié" gdb avec ce qui execute votre programme (qemu-arm : étape précédante). Pour cela nous allons le faire par le biais des ports de communications de votre machine (option -g de qemu-arm). Il faut bien respecter l'ordre des commandes suivante :

`qemu-arm -L /usr/arm-linux-gnueabi/ -g 1234 ./<output_nameFile> <arg0> ... <argN>`

Ainsi qemu attend une connexion au port 1234 de votre machine avant toute execution du programme output_nameFile et de ses arguments arg0 ... argN. Il nous manque plus qu'a lancé gdb et de lui dire d'écouter le port 1234.

`gdb-multiarch -ex "target remote 1234"`

Ensuite vous pouvez continuer avec l'utilisation habituelle de gdb que vous avez (comme le debug d'un programme C : b main, run, next, step, etc).

En petit bonus à la place de "-ex"target remote 1234"" de la commande ci-dessus, vous pouvez créer un fichier qui contiendra toute les commandes gdb (dont le target remote 1234)qui seront éxécutés au lancement de gdb (c.f arm_debug.gdb -> je vous laisse vous renseigner sur les différentes commandes qui y figurent). Ainsi la nouvelle commande pour lancer gdb-multiarch sera :

`gdb-multiarch --command=PATH/TO/<debug_arm.gdb>`

Ressources supplémentaires

Erreur : Command 'arm-linux-gnueabi-gcc' not found

Pour résoudre cela nous allons chercher tout les fichiers executables qui appartiennent au pacquet gcc-arm-linux-gnueabi :

`dpkg -L gcc-arm-linux-gnueabi | grep -E '/s?bin/'`

Cette commande nous donne (normalement) une liste. L'executable qui nous ai nécessaire doit contenir uniquement gcc. Une fois trouvé vous remplacer arm-linux-gnueabi-gcc par celui que vous avez trouvé.

Erreur : qemu-arm: Could not open '/lib/ld-linux.so.3': No such file or directory

Pour cette erreur nous devons trouver où est ce qu'a était rangé la bibliothèque `arm-linux-gnueabi` par le système. La commande suivante va nous aider :

`dpkg -L binutils-arm-linux-gnueabi | grep "arm-linux-gnueabi/bin/ar"`

`retourne (sur ma machine): /usr/arm-linux-gnueabi/bin/ar`

Ceci est sensé nous retourner le chemin vers l'executable "ar" appartenant à la bibliothèque rechercher, qui est unique dans le paquet binutils-arm-linux-gnueabi. Ainsi nous avons notre chemin : "/usr/arm-linux-gnueabi". Il manque plus qu'à remplacer dans la commande d'éxécution d'un programme (apres l'option -L de qemu-arm).

Dernière mise à jour : [25/03/2026]