OS

Par fiakx

1 Il y a un début à tout, non ?

1.1 Cross Compiler

1.1.1 Contexte

Quand tu développes un système d'exploitation, tu travailles généralement sur une machine (qu'on appellera **l'hôte**) qui a déjà un OS installé (comme Linux, Windows, ou macOS). Cependant, ton objectif est de créer un nouvel OS qui fonctionnera sur une autre machine (qu'on appellera ici **cible**), qui peut avoir une architecture matérielle différente (par exemple, x86, ARM, etc.).

Le problème est que le compilateur présent sur ta machine hôte est configuré pour produire des programmes qui fonctionnent sur cette machine hôte, pas sur la machine cible. C'est là qu'intervient la notion de **cross-compilation**.

1.1.2 Cross-Compilation

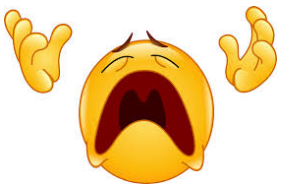
Un **cross-compilateur** est un compilateur qui fonctionne sur une plateforme (hôte) mais qui produit des exécutables pour une autre plateforme (cible). Par exemple, si tu développes sur un PC x86 sous Linux, mais que tu veux créer un OS pour une machine ARM, tu as besoin d'un cross-compilateur qui tourne sur x86 mais qui génère du code pour ARM.

1.1.3 Pourquoi est-ce nécessaire ?

La cross-compilation est essentielle pour plusieurs raisons :

— **Architecture différente** : Si la cible a une architecture matérielle différente de celle de l'hôte, le compilateur natif ne peut pas produire de code exécutable pour cette architecture.
— **Environnement de développement** : Tu veux pouvoir développer et tester ton OS sur une machine qui a déjà un OS fonctionnel, sans avoir à tout réécrire pour la cible.
— **Efficacité** : Utiliser un cross-compilateur te permet de générer rapidement des binaires pour la cible sans avoir à installer un environnement de développement complet sur la cible.

1.1.4 Oui, mais comment on installe tout ça ?



Télécharger touskilfaut pour le compilateur

Tu auras besoin des sources de GCC (GNU Compiler Collection) et de binutils (un ensemble d'outils pour manipuler les binaires).

Configurer le cross-compilateur

Tu dois configurer GCC pour qu'il sache qu'il doit produire du code pour la cible et non pour l'hôte. Cela se fait généralement en spécifiant des options de configuration lors de la compilation de GCC.

Par exemple, si tu veux compiler pour une architecture ARM, tu utiliserais une commande comme :

```
./configure --target=arm-none-eabi --prefix=/usr/local/cross
```

— `--target` spécifie l'architecture cible (ici, `arm-none-eabi`).
— `--prefix` indique où installer le cross-compilateur (ici, `/usr/local/cross`).

Compiler et installer

Une fois configuré, tu compiles et installes le cross-compilateur. Cela peut prendre un certain temps car GCC est un gros projet.

Utiliser le cross-compilateur

Une fois installé, tu peux utiliser ce compilateur pour générer des exécutables pour ta cible. Par exemple, si tu as un fichier C `hello.c`, tu peux le compiler avec :

```
arm-none-eabi-gcc -o hello hello.c
```

Cela produira un exécutable `hello` qui fonctionnera sur une machine ARM.