0compiler : Chroot Build

Table of Contents

Cet article explique comment créer et configurer un environnement de construction **0compile**, offrant les avantages suivants :

- La construction n'affecte pas l'environnement hôte, et l'hôte n'influence pas la construction.
- Les builds peuvent cibler une architecture différente, comme la construction de packages x86 à partir d'un hôte x64.
- Le bac à sable de build est plus petit qu'une machine virtuelle et le téléchargement est généralement plus petit.

Le système de construction (chroot) est fourni avec des outils de développement tels que **gcc** et **make**, et la commande **0launch**.

Linux From Scratch

Utiliser une toolchain **L**inux **F**rom **S**cratch (**LFS**), en utilisant uniquement des archives sources en chroot

Le site https://github.com/emmett1/lfs-scripts propose des scripts pour automatiser la compilation multilib LFS + livecd. Cette version LFS utilise les pkgutils de CRUX pour gérer les packages et le générateur initramfs de Venom Linux pour livecd initramfs.
Il existe 3 scripts:
- **01-Toolchan** script pour construire la toolchain nécessaire pour créer des lfs de base (obligatoire), ce script doit être exécuté en tant qu'utilisateur normal, ce script peut être repris, il suffit de réexécuter le script pour continuer là où vous l'a laissé
- **02-base**: script pour construire le système lfs (des packages supplémentaires sont ajoutés dans cette base lfs, tels que noyau Linux, wget, dhcpcd, wpa_suppliant, mkinitramfs, syslinux) de base, ce script doit être exécuté en tant que root, tout le paquet est construit en utilisant le système de port (pkgutils), on peut créer ses propres ports, ce script peut être repris, il suffit de réexécuter le script pour continuer là où on l'a laissé, les packages créés peuvent être réutilisés
- **03-mkiso**: script pour construire lfs livecd iso, ce script doit être exécuté en tant que root

Fondamentalement, on a juste besoin d'exécuter tous ces 3 scripts sans autre commande pour que le système LFS soit construit, y compris l'ISO, en exécutant simplement `$ ./01-toolchain && sudo ./02-base && sudo ./03-mkiso`

Pour construire la toolchain on peut utiliser une distribution Linux comme hôte ou récupérer un livecd de n'importe quelle distribution:
- Préparer une partition pour LFS et la monter sur /mnt/lfs (on peut changer le répertoire de construction de LFS dans le fichier de configuration).
- Modifier éventuellement le fichier de configuration en fonction des besoins.
- Exécuter le script `01-toolchain` pour créer une toolchain temporaire.
- Exécuter le script `02-base` pour créer le système LFS de base.
-Exécuter éventuellement le script `03-mkiso` pour créer l'iso (on peut tester l'iso à l'aide de qemu en exécutant `./run_qemu <fichier iso>`).
- Exécuter `./enter-chroot` pour entrer dans l'environnement chroot pour configurer le système.

Le site https://intestinate.com/pilfs/ propose des images précompilées de la toolchain LFS, on peut utiliser une de ce images comme base pour un projet ou comme environnement de construction dans lequel construire une distribution en suivant le guide PiLFS.

Configuration du chroot

Entrer dans le chroot :

```
export LFS=/mnt/lfs
https://intestinate.com/pilfs/images.html

sudo mount -v --bind /dev $LFS/dev

sudo mount -vt devpts devpts $LFS/dev/pts
sudo mount -vt tmpfs shm $LFS/dev/shm
sudo mount -vt proc proc $LFS/proc
sudo mount -vt sysfs sysfs $LFS/sys

sudo mkdir -pv $LFS/build
sudo mount -v --bind ./build $LFS/build

$ sudo chroot $LFS /usr/bin/env -i \
    HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /bin/bash --login
```

Installation de Zero

Les paquets BLFS suivants doivent être installés:

- OpenSSL (Python dependency)
- Python
- GnuPG
- PCRE (Glib dependency)
- GLib (PyGObject dependency)
- PyGObject

  Télécharger le code source **0install** pour les versions publiées à partir de la page GitHub Releases.

On peut également obtenir la dernière version de développement à l'aide de Git : `git clone https://github.com/0install/0install.git`

Pour installer pour tous les utilisateurs du système (avec accès root) :

```
$ make
$ sudo make install
```

Pour installer uniquement pour l'utilisateur actuel (sans accès root) :

```
$ make && make install_home
$ export PATH=$HOME/bin:$PATH
```

Configuration de 0compile

Maintenant, on a un chroot avec **0launch**, et on peut ajouter **0compile** :

```
CMD="0compile"
URI="https://apps.0install.net/0install/0compile.xml"

$ yes Y | /usr/bin/0launch -cd $URI
$ 0alias -d /usr/bin $CMD $URI
```

Selon la version du système d'exploitation et de Python, il faudra peut-être utiliser une ancienne version de 0compile.

Construction avec 0compile

Télécharger le code source, en mode console :

```
$ 0launch -cd -s http://www.example.com/interfaces/foo.xml
```

Ensuite, configurer le sous-répertoire build en utilisant le flux source :

```
$ cd /build
$ 0compile setup http://www.example.com/interfaces/foo.xml foo
```

Ensuite, procéder à la construction du binaire à partir de la source :

```
$ cd foo
$ 0compile build
```

Enfin, dire à **0compile** de préparer le flux/l'archive binaire :

```
$ 0compile publish http://www.example.com/implementations
```

Après avoir quitté le chroot, on peut trouver les résultats dans build/foo.

Publication des résultats

Le nouveau flux binaire est maintenant prêt à être fusionné avec le flux source, signé (à l'aide de `0publish --xmlsign`) et publié avec les archives.

Puisqu'on a utilisé un nouveau chroot propre pour construire le binaire, on peut être raisonnablement sûrs que toutes les dépendances sont incluses dans le flux source.