Système de fichiers racine

M2 SETI B4 / MS SE SE758

Guillaume Duc
guillaume.duc@telecom-paris.fr
2021–2022

Plan

Compilateur croisé

Cross compiler

- Un compilateur croisé (*cross compiler*) est un compilateur qui s'exécute sur une plate-forme A et qui produit du code exécutable pour une plate-forme B
- Dans le cas le plus fréquent, la différence entre les deux plate-formes se situe au niveau des architectures de processeurs (exemple : compiler du code destiné à s'exécuter sur un processeur ARM 32 bits depuis une machine `x86_64`)
- La différence peut aussi se situer au niveau des systèmes d'exploitation (exemple : produire un exécutable pour Windows à partir d'un compilateur tournant sous Linux)

Compilateur croisé
Utilité

- Les compilateurs croisés sont très utilisés lors du développement pour l'embarqué
- En effet, il est souvent plus agréable, rapide, etc. de développer et de compiler sur une machine puissante (souvent `x86_64`) alors que le système embarqué que l'on vise est petit et contraint en ressources (souvent à base d'architecture ARM, MIPS...)

Compilateur croisé
Création

- La génération d'un compilateur croisé (par exemple `gcc`) n'est pas très simple
- Outre les bibliothèques et outils utilisés directement par `gcc`, il a besoin :
  - GNU `binutils` (notamment `as`, `ld`) compilés pour la plate-forme cible
  - De la bibliothèque standard du C (notamment des fichiers d'en-tête) qui sera utilisée sur la plate-forme cible
- De nombreux scripts existent pour faciliter la génération d'un compilateur croisé (exemple : `buildroot`)

Compilateur croisé
Création

- De plus, des compilateurs croisés prêts à l'emploi pour les cas les plus fréquents sont disponibles
  - soit dans les distributions Linux classiques (exemple : chaîne `arm-none-eabi-gcc` disponible depuis le paquet du même nom sous Debian ou Arch Linux)
  - soit par exemple depuis Linaro (exemple : chaîne `arm-linux-gnueabihf`)

Bibliothèque standard du C

- La bibliothèque standard du C contient les fichiers d'en-tête (`.h`) et l'implémentation des fonctions standard du C (`strlen`, `malloc`, `printf`, etc.)
- Son contenu est standardisé, à la fois par le standard ANSI/ISO C, mais également dans les spécifications POSIX
- Elle est nécessaire pour la compilation et l'exécution des programmes développés en C
- Elle peut être liée statiquement ou dynamiquement aux exécutables produits lors de l'édition des liens

Bibliothèque standard du C

- Outre les fonctions de base (comme la manipulation des chaînes de caractères), elle joue un rôle essentiel dans l'interface avec le système d'exploitation (en fournissant une interface pour réaliser les appels systèmes essentiels, comme par exemple : `fork`, `open`, `read`...)
- Dans les systèmes Unix, elle n'est pas simplement utilisée que par les programmes en C mais fait partie *de facto* de l'interface entre l'espace utilisateur et le noyau
- Elle est donc, sauf cas particuliers, indispensable

Bibliothèque standard du C

GNU C Library (glibc)

- L'implémentation la plus connue et la plus utilisée dans le monde Linux est la *GNU C Library* (aussi connue sous le nom `glibc`)
- Elle est développée depuis 1987 (première version) dans le cadre du projet GNU et publiée sous licence LGPL
- Elle est compatible avec de nombreux standards, comme par exemple : ISO C11, POSIX.1-2008 et IEEE 754-2008
- Son plus gros inconvénient est sa taille (plusieurs Mo), ce qui peut être rédhibitoire dans le domaine de l'embarqué

Bibliothèque standard du C

Petit rappel sur les bibliothèques partagés

- Lors de la production d'un exécutable, l'éditeur des liens (par exemple `ld`) peut intégrer le code des bibliothèques utilisées (et notamment celui de la bibliothèque standard du C) directement dans l'exécutable produit (*liaison statique*)
  - L'exécutable contient alors tout le code nécessaire pour l'exécution et ne dépend donc pas de ce qui est installé sur le système cible
  - L'exécutable est plus gros
  - La mise à jour en cas de découverte d'un bug dans une bibliothèque impose la régénération de l'exécutable
  - Si deux processus utilisent la même bibliothèque, le code de celle-ci sera présent deux fois en mémoir

Bibliothèque standard du C

Petit rappel sur les bibliothèques partagés

- L'éditeur des liens peut également faire une *liaison dynamique* avec une bibliothèque partagée
  - Le code de la bibliothèque n'est pas intégré à l'exécutable
  - À l'exécution, l'éditeur de liens dynamique (`ld-linux.so`) va chercher les bibliothèques partagées utilisées, les charger en mémoire et finaliser l'édition des liens du programme
  - Si deux processus partagent une même bibliothèque, son code n'est chargé qu'une seule fois en mémoire, dans une section mémoire partagée
  - En cas de bug, il suffit simplement de mettre à jour la bibliothèque sur le système cible pour que les programmes utilisent cette nouvelle version
  - L'exécutable est plus petit

Bibliothèque standard du C

Alternatives

- Un certain nombre de bibliothèques se proposent de remplacer la `glibc` dans le domaine de l'embarqué avec des tailles beaucoup plus petites
- On peut notamment citer :
  - Newlib
  - dietlibc
  - uClibc / uClibc-ng

Plan

Organisation

- L'organisation de la racine des systèmes de fichiers sous Linux est standardisée (*Filesystem Hierarchy Standard*, `https://refspecs.linuxfoundation.org/fhs.shtml`)

/bin

- Commandes essentielles utilisables par tous les utilisateurs, notamment si aucun autre système de fichiers n'est monté
- Exemples : `cat`, `cp`, `dd`, `ls`, `login`, `mount`, `sh`, `su`...

/boot

- Fichiers utilisés par un éventuel *bootloader*
- Typiquement, on peut y trouver les noyaux compilés, les images mémoires initiales (`initramfs`) et les fichiers de configuration du bootloader (par exemple GRUB)

/dev

- Fichiers spéciaux représentant les périphériques du système
- Trois paramètres sont associés à chacun de ces fichiers
  - Un mode (`b` pour bloc, `c` pour caractère)
  - Un numéro *majeur*
  - Un numéro *mineur*
- À l'aide de ces trois paramètres (le nom même du fichier n'a pas d'importance en général), le noyau fait l'association avec un périphérique (présent ou non dans le système)
- Les applications réalisent des opérations sur un périphérique en utilisant des appels systèmes sur le fichier correspondant (`open`, `read`, `write`, `ioctl`...)

/dev
Exemples

- `/dev/console` (caractère, majeur 5, mineur 1) représente la console principale
- `/dev/hidraw0` (caractère, majeur 240, mineur 0) représente le premier périphérique HID (*Human Interface Device*)
- `/dev/sda` (bloc, majeur 8, mineur 0) représente le premier disque dur SATA/SCSI
- `/dev/sda1` (bloc, majeur 8, mineur 1) représente la première partition du premier disque dur SATA/SCSI
- `/dev/ttyS0` (caractère, majeur 4, mineur 64) représente le premier terminal série
- `/dev/zero` (caractère, majeur 1, mineur 5) fichier spécial dans lequel on ne lit que des zéros et dans lequel les écritures disparaissent

/dev
Peuplement

- Ces fichiers spéciaux peuvent être créés (n'importe où, pas nécessairement dans `/dev`) à l'aide de la commande `mknod` en fournissant le mode, le numéro majeur et le numéro mineur (nécessite d'être super-utilisateur)
- Historiquement, un script (`MAKEDEV`) dans `/dev` permettait de créer tous les fichiers spéciaux possibles (même s'ils ne correspondaient pas à un périphérique réellement présent) en appelant `mknod`
- Cette méthode est toujours employée dans certains systèmes embarqués (soit par l'intermédiaire d'un script, soit manuellement à la création du système de fichiers)

/dev
Peuplement

- Des mécanismes permettent de créer automatiquement les fichiers dans `/dev` en fonction des périphériques présents réellement sur le système
- Avant le noyau 2.6, il s'agissait de `devfsd`
- Depuis, ce rôle est essentiellement rempli par `udev` et/ou `devtmpfs`

/dev

devtmpfs

- `devtmpfs` est un type de système de fichiers particulier, généralement monté dans `/dev`
- C'est un système de fichiers mémoire (comme `tmpfs`) spécial et qui est automatiquement rempli par le noyau avec les fichiers spéciaux des périphériques présents sur le système
- Pour l'utiliser il suffit de le monter : `mount -t devtmpfs devtmpfs /dev`
- Une option du noyau (`CONFIG_DEVTMPFS_MOUNT`) permet même d'indiquer au noyau de le monter automatiquement juste après le système de fichiers racine (sauf dans le cas d'une image `initramfs`)

/dev

udev

- `udev` est un démon tournant dans l'espace utilisateur
- Il écoute (par l'intermédiaire d'une *socket*) les messages `uevents` envoyés par le noyau lors d'événements (détection d'un nouveau périphérique, disparition d'un périphérique...)
- Il va ensuite réagir à ces événements sur la base d'un ensemble de règles configurables
  - Création / suppression d'un fichier spécial (si ce n'est pas déjà fait par `devtmpfs`)
  - Attribution de permissions particulières sur ces fichiers
  - Exécution d'une commande particulière (configuration du périphérique, chargement d'un micro-code, etc.)...

/dev/pts

- On trouve également classiquement dans `/dev`, un répertoire `/dev/pts` dans lequel est monté un système de fichiers particulier (`devpts`)
- Il est utilisé pour le contrôle des pseudo-terminaux

/etc

- `/etc` (historiquement pour *etcetera*, tout ce qui ne va pas ailleurs) contient les fichiers de configuration des différentes applications
- Fichiers classiques
  - `fstab` : systèmes de fichiers à monter
  - `group` : liste des groupes
  - `host.conf`, `resolv.conf` : configuration du résolveur de noms
  - `hosts` : configuration statique des noms d'hôtes
  - `inittab` : configuration d'`init`
  - `passwd` : liste des utilisateurs (et anciennement mots de passe)

/etc

- Fichiers classiques (suite)
  - `protocols` : liste des numéros des protocoles réseaux
  - `services` : liste des ports TCP/UDP des différents services
  - `shadow` : liste des mots de passe des utilisateurs
  - `shells` : liste des interpréteurs de commandes valides

/home

- Répertoires personnels des utilisateurs

/lib

- Bibliothèques partagées essentielles et modules du noyau
  - `.so` : bibliothèques partagées. Exemple, pour une bibliothèque `bbb`, on a classiquement
    - `libbbb.so.x.y.z` : le fichier `.so` contenant la bibliothèque partagée
    - `libbbb.so.x` : lien symbolique vers `libbbb.so.x.y.z` (utilisé par l'éditeur de liens dynamique lors de l'exécution, `x` étant le numéro de version majeur, ne changeant qu'en cas d'incompatibilité)
    - `libbbb.so` : lien symbolique vers `libbbb.so.x.y.z` (utilisé par l'éditeur de liens lors de la compilation d'un programme)
  - `.a` : fichiers utilisés pour la liaison statique (archive contenant des `.o`)
- `libc.so.6` : Bibliothèque standard du C

/lib

- `ld-linux.so.*` : Éditeur de liens dynamique
- `/lib/modules` : Modules du noyau (dans des sous répertoires portant le numéro de version du noyau)

`/libxxx` (exemple `lib64`)

- Présent si des bibliothèques partagées sous un format différent sont nécessaires (on peut ainsi mixer sur un système 64 bits des bibliothèques 32 et 64 bits)

/media

- Contient des sous-répertoires qui sont des points de montage pour des médias amovibles
- Exemple
  - `/media/cdrom`
- C'est une convention assez récente (avant : `/cdrom` ou `/mnt/cdrom`) qui n'est pas universellement adoptée

/opt

- Logiciels supplémentaires, souvent en provenance d'autres sources que le système de packaging de la distribution

/root

- Répertoire personnel de l'utilisateur `root`

/run

- Répertoire servant à stocker des informations utiles sur les programmes en cours de fonctionnement
- En général remis à zéro à chaque démarrage (peut par exemple être mis dans un `tmpfs`)
- Remplaçant de `/var/run` (`/var/run` est maintenant le plus souvent un lien symbolique vers `/run`)
- Si une application a besoin d'y stocker plusieurs fichiers, la convention veut qu'il crée un sous-répertoire à son nom

/run

- On y trouve notamment
  - `xxx.pid` : fichier contenant le PID du programme `xxx`
  - Des sockets ou des tubes nommés servant à communiquer avec une application ou aux différentes parties d'une application à communiquer ensemble

`/sbin` et `/usr/sbin`

- Commandes utilisées pour l'administration du système (généralement réservées à l'utilisateur `root`)
- `/sbin` : celles nécessaires pour le démarrage et la réparation éventuelle du système
- `/usr/sbin` : les autres

/srv

- Emplacement pour les données servies par le système
- Exemple : pour un serveur HTTP, les fichiers servis pourraient être dans `/srv/www` ou `/srv/http`

/tmp

- Fichiers temporaires
- Aucune garantie de conservation entre deux invocation d'un programme
- En général stocké sur un `tmpfs`
- Accessible en écriture pour tous les utilisateurs + bit *sticky* (`t`) pour empêcher un utilisateur de supprimer un fichier appartenant à un autre utilisateur (ce qui est normalement permis si un utilisateur à le droit d'écriture sur un répertoire)

/usr

- `/usr/bin` : principaux exécutables
- `/usr/include` : fichiers d'en-tête pour le langage C
- `/usr/lib` : bibliothèques partagées (souvent `/lib` et `/usr/lib` pointent vers le même répertoire)
- `/usr/local` : hiérarchie pour les logiciels installés localement (souvent avec un `make install`), on y retrouve la même hiérarchie que dans `/usr`
- `/usr/sbin` : commandes systèmes non essentielles

/usr

- `/usr/share` : données indépendantes de l'architecture (souvent un répertoire par application + `man` pour les pages de manuel + `doc` pour la documentation)

/var

- Données variables des applications et du système
- `/var/cache` : cache des applications
- `/var/lib` : données persistantes des applications entre plusieurs appels
- `/var/lock` : fichiers verrou
- `/var/log` : journaux
- `/var/mail` : boites mails des utilisateurs
- `/var/opt` : hiérarchie pour les logiciels installés dans `/opt`

/var

- `/var/run` : voir `/run`
- `/var/spool` : données en attente de traitement (exemple : mails en attente de livraison, impressions en attente...)
- `/var/tmp` : fichiers temporaires préservés entre redémarrages

/proc

- Informations sur le noyau et les processus
- Type de système de fichiers spécial `proc`
- Contient des fichiers créés automatiquement par le noyau
- Exemples
  - `cmdline` : arguments passés au noyau lors du démarrage
  - `cpuinfo` : informations sur les processeurs
  - `mounts` : liste des systèmes de fichiers montés
  - `config.gz` : configuration du noyau
  - `interrupts` : information sur les interruptions
  - `modules` : liste des modules chargés

/proc/nnn

- Informations sur le processus de PID `nnn`
- Exemples
  - `cwd` : lien vers le répertoire courant d'exécution
  - `exe` : lien vers l'exécutable
  - `cmdline` : arguments
  - `environ` : variables d'environnement
  - `pagemap` : table des pages du processus (voir `Documentation/vm/pagemap.txt` pour son fonctionnement)
- `/proc/self` est un lien symbolique spécial pointant vers le répertoire `/proc/nnn` du processus courant

/sys

- Informations sur le noyau (notamment sur les périphériques et les pilotes de périphériques)
- Type de système de fichiers spécial `sysfs`
- Contient des fichiers créés automatiquement par le noyau
- Exemples
  - `/sys/bus/i2c/devices/` : liste des périphériques `i2c` (avec pour chacun de nombreuses informations : nom, pilote en charge, branche du DTB correspondante...)
  - `/sys/bus/i2c/drivers/` : liste des pilotes de périphériques enregistrés auprès du système `i2c`

Plan

Format des systèmes de fichiers

- Les différents répertoires et fichiers que nous venons de voir doivent être stockés dans un (ou éventuellement plusieurs) système de fichiers
- Il existe de nombreux formats pour ces systèmes de fichiers
  - Formats classiques : `ext2`, `ext3`, `ext4`, `btrfs`, `ZFS`, `FAT`...
  - Formats conçus spécifiquement pour les mémoires flash ou les systèmes embarqués : `cramfs`, `JFFS2`, `SquashFS`, `YAFFS2`, `F2FS`...
  - Formats mémoire : `ramfs`, `tmpfs`...
  - Formats spéciaux : `sysfs`, `proc`...

Particularités dans le monde de l'embarqué

- Dans de nombreux cas, dans le monde de l'embarqué, les données vont être stockées sur des puces de mémoire flash
- Une particularité importante de ces mémoires est le relativement faible nombre d'écritures autorisées sur une cellule donnée avant sa destruction définitive
- Certains contrôleurs intègrent en matériel un mécanisme de répartition de l'usure (*wear leveling*) permettant de déplacer automatiquement les secteurs souvent écrits régulièrement pour retarder l'apparition des problèmes

Systèmes de fichiers en lecture seule

- `cramfs`
  - Compressé
  - Taille maximale du système environ 256 Mio
  - Méta-données simplifiées (identifiant de groupe stocké sur 8 bits, pas de datation...)
- `SquashFS`
  - Compressé
  - Plus récent mais plus gourmand en mémoire que `cramfs`
  - Taille maximale du système environ 16 Eio
- Peuvent être utilisés pour stocker le squelette de l'arborescence ainsi que tous les fichiers et exécutables qui n'ont pas besoin d'être modifiés, le reste (`/var`, `/run`...) pouvant être crée sur un système de fichiers mémoire

Systèmes de fichiers spécialisés pour les mémoires flash

- Gérant la répartition de l'usure en logiciel
  - `JFFS2` (*Journalling Flash File System version 2*)
  - `YAFFS1/2` (*Yet Another Flash File System*) : spécialisé pour les mémoires NAND
- Ne gérant pas la répartition de l'usure (se basent sur une *Flash translation layer*)
  - `F2FS` (*Flash-Friendly File System*)

Systèmes de fichiers généralistes

- `ext2/3/4` (*second/third/fourth extended file system*) : systèmes de fichiers utilisés par la quasi-totalité des distributions Linux, `ext2` est intégré au noyau Linux depuis la version 0.99, `ext3` ajoute le support de la journalisation (pour plus de fiabilité), `ext4` apporte une augmentation des limites et des optimisations
- Systèmes avancés (scalables, supportant plusieurs supports de stockage avec éventuellement de la redondance, etc.)
  - `btrfs` (*b-tree file system*)
  - `ZFS` (*Zettabyte file system*) : developpé par Sun/Oracle, licence incompatible avec celle du noyau, requiert beaucoup de mémoire vive

Plan

Récapitulatif des besoins

- Cross-compilateur
- Bibliothèque standard du C
- Bootloader
- Noyau
- Arbre des périphérique
- Image mémoire initiale (facultatif)
- Système de fichiers racine avec arborescence classique et différents utilitaires + programmes

Problèmes

- Long à faire à la main
- Dépendances / conflits entre les versions des outils
- Mise à jour ?

Automatisation

- Générateur de distribution Linux
  - Yocto Project : `https://www.yoctoproject.org`
  - Buildroot : `https://buildroot.org`
  - OpenEmbedded : `http://www.openembedded.org`
- Les deux plus utilisés couramment sont *Yocto* (plus récent, plus complet mais plus complexe a prendre en main) et *Buildroot* (plus ancien, apprentissage plus simple)
- Nous allons voir le fonctionnement de base de Buildroot en TP