Linux From Scratch embarqué

en 40 minutes !

Michael Opdenacker
Free Electrons
http://free-electrons.com/

Traduction française par Guillaume Lelarge

```
nada + 40 min =
```

Créé avec OpenOffice.org 2.x

Droits de copie

creative commons
COMMONS DEED

Attribution – ShareAlike 2.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions

Attribution. You must give the original author credit.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: http://creativecommons.org/licenses/by-sa/2.0/legalcode

Sources du document, mises à jour et traductions:
http://free-electrons.com/articles/elfs

Corrections, suggestions, améliorations et traductions sont les bienvenues!

Plus facile à lire avec...

Ce document est le plus facile à lire avec un lecteur PDF récent ou avec OpenOffice.org lui-même! Vous pouvez:

- Utiliser les hyperliens internes ou externes.
  Ainsi, n'hésitez pas à cliquer sur ces liens!
- Trouver facilement des pages grâce à la recherche automatique.
- Utiliser les miniatures de pages pour naviguer rapidement dans le document.

Si vous lisez une copie papier ou HTML, vous feriez mieux de récupérer une copie au format PDF ou OpenOffice.org sur http://free-electrons.com/articles/elfs!

Remerciements

- À ceux qui ont envoyé des corrections ou des suggestions:
  Florent Peyraud, Fabrice Menard
- Aux traducteurs et traductrices:
  Guillaume Lelarge (français)

Buts du tutoriel

Construire un petit système embarqué à partir de rien, en 40 minutes

- Configuration et compilation du noyau Linux
- Création du système de fichiers racine
- Compilation et installation de Busybox
- Création des fichiers spéciaux pour les périphériques
- Scripts de démarrage du système : systèmes de fichier virtuels, réseau
- Configuration d'une interface HTTP simple

Vous montrer à quel point cela est simple !

Approche du plus vers le moins

Approche du plus vers le moins pour construire un système embarqué

- Commencer à partir d'un environnement de bureau complet sous GNU/Linux (Debian, Fedora...) et supprimer tous les éléments inutiles.
- Travail très complexe : besoin de vérifier un très grand nombre de fichiers et de paquetages. Besoin de comprendre l'(in)utilité de chaque fichier avant de le supprimer.
- Conserver des scripts et des fichiers de configuration inutilement complexes.
- Le résultat final est quand même trop gros car les outils et les bibliothèques standards sont utilisés. De plus, beaucoup de bibliothèques partagées sont nécessaires.

Approche du moins vers le plus

Approche du moins vers le plus pour construire un système embarqué

- Commencer avec un système de fichiers minimal, voire vide, en ajoutant seulement les éléments qui vous sont nécessaires.
- Bien plus facile à réaliser ! Vous passez du temps sur ce dont vous avez besoin.
- Plus facile à contrôler et à maintenir : vous développez une compréhension des outils que vous utilisez.
- Vous avez seulement besoin de scripts de configuration basiques.
- Le résultat final peut être extrêmement petit, d'autant plus que vous utilisez des outils légers.

Outils utilisés dans ce tutoriel

Diapositives explicatives à montrer lors de la compilation

qemu

http://qemu.org
Émulateur rapide de processeur
utilisant un traducteur dynamique portable.

Deux modes opérationnels

- Émulation d'un système complet : processeur et périphériques divers
  Support de `x86`, `x86_64`, `ppc`
- Émulation du mode utilisateur (seulement sur un hôte Linux) : peut exécuter des applications compilées pour d'autres CPU.
  Support de `x86`, `ppc`, `arm`, `sparc`

Exemples avec qemu

Émulation utilisateur

- Facile d'exécuter Busybox pour `arm` sur `i386` GNU / Linux :
```
qemu-arm -L /usr/local/arm/3.3.2 \
/home/bart/arm/busybox-1.00-pre8/busybox ls
```
- `-L` : chemin des binaires de la bibliothèque C cible (ici chemin de la chaîne de cross-compilation)

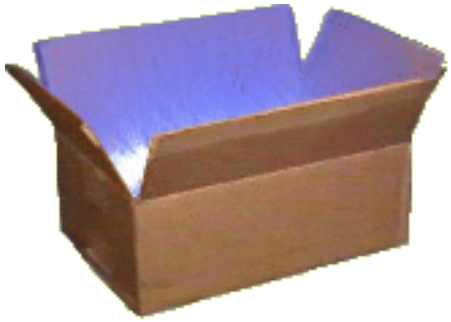
Émulation système

- Encore plus facile à exécuter :
```
qemu linux.img
```
- `linux.img` : image complète de la partition incluant le noyau
  Plein d'images pour les systèmes d'exploitation libres sur http://free.oszoo.org !

Bysybox: boîte à outils générique

http://www.busybox.net/ de Codepoet Consulting

- La plupart des outils Unix en ligne de commande avec un seul exécutable ! Inclut même un serveur web !
- Fait moins d'1 Mo (compilé statiquement avec glibc)
  moins de 500 Ko (compilé statiquement avec uClibc)
- Configuration aisée des fonctionnalités à inclure
- Le meilleur choix pour
  - Initrds avec des scripts complexes
  - Tout système embarqué !



Commandes de busybox !

addgroup, adduser, adjtimex, ar, arping, ash, awk, basename, bunzip2, bzcat, cal, cat, chgrp, chmod, chown, chroot, chvt, clear, cmp, cp, cpio, crond, crontab, cut, date, dc, dd, deallocvt, delgroup, deluser, devfsd, df, dirname, dmesg, dos2unix, dpkg, dpkg-deb, du, dumpkmap, dumpleases, echo, egrep, env, expr, false, fbset, fdflush, fdformat, fdisk, fgrep, find, fold, free, freeramdisk, fsck.minix, ftpget, ftpput, getopt, getty, grep, gunzip, gzip, halt, hdparm, head, hexdump, hostid, hostname, **httpd**, hush, hwclock, id, ifconfig, ifdown, ifup, inetd, init, insmod, install, ip, ipaddr, ipcalc, iplink, iproute, iptunnel, kill, killall, klogd, lash, last, length, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, ls, lsmod, makedevs, md5sum, mesg, mkdir, mkfifo, mkfs.minix, mknod, mkswap, mktemp, modprobe, more, mount, msh, mt, mv, nameif, nc, netstat, nslookup, od, openvt, passwd, patch, pidof, ping, ping6, pipe_progress, pivot_root, poweroff, printf, ps, pwd, rdate, readlink, realpath, reboot, renice, reset, rm, rmdir, rmmod, route, rpm, rpm2cpio, run-parts, rx, sed, seq, setkeycodes, sha1sum, sleep, sort, start-stop-daemon, strings, stty, su, sulogin, swapoff, swapon, sync, sysctl, syslogd, tail, tar, tee, telnet, telnetd, test, tftp, time, top, touch, tr, traceroute, true, tty, udhcpc, udhcpd, umount, uname, uncompress, uniq, unix2dos, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat

glibc

http://www.gnu.org/software/libc/

- Bibliothèque C du projet GNU
- Conçue pour la performance, le respect des standards et la portabilité
- Trouvée sur tous les systèmes GNU / Linux
- Assez gros sur des petits systèmes embarqués : à peu près 1,7 Mo sur les iPAQs Linux (`libc` : 1,2 Mo, `libm` : 500 Ko)
- Taille du programme exemple “hello world”: 12 Ko (lié dynamiquement), 350 Ko (lié statiquement)

uClibc

http://www.uclibc.org/ de CodePoet Consulting

- Bibliothèque C légère pour les petits systèmes embarqués, avec néanmoins beaucoup de fonctionnalités
- Debian Woody a été entièrement porté sur cette bibliothèque... Vous êtes certain qu'elle satisfera tous vos besoins !
- Taille exemple (`arm`) : approx. 400 Ko (`libuClibc` : 300 Ko, `libm` : 55Ko)
- Taille du programme exemple “hello world” : 2 Ko (lié dynamiquement), 18 Ko (lié statiquement).

Interface noyau en espace utilisateur

Quelques exemples :

- `/proc/cpuinfo` : informations sur le processeur
- `/proc/meminfo` : statut de la mémoire
- `/proc/version` : informations sur la version et la construction
- `/proc/cmdline` : ligne de commande du noyau
- `/proc/<pid>/environ` : environnement d'appel
- `/proc/<pid>/cmdline` : ligne de commande du processus

... et bien plus encore ! Détails complets dans les sources du noyau :
`Documentation/filesystems/proc.txt`

Ce que nous avons réalisé

Compilation du noyau Linux

- Récupération des sources sur http://kernel.org
- Initialisation d'une configuration minimale du noyau :
  `make allnoconfig`
- Ajout de paramètres spécifiques au système embarqué :
  `make xconfig` ou `make menuconfig`
  (the kernel configuration file that we use can be found on
  http://free-electrons.com/doc/embedded_lfs/linux-2.6.15.config)
- Compilation :
  `make`
- Résultat : `image du noyau compressé`
  `arch/i386/boot/bzImage`

Création d'un système de fichiers racine

- Création d'un fichier vide de `320 Ko` :

```
dd if=/dev/zero of=rootfs.img bs=320k count=1
```

- Formatage d'un fichier en ext2 :

```
mkfs.ext2 -i 1024 -F rootfs.img
```

une inode pour 1024 octets -> 320 fichiers

au lieu d'une inode pour 4096 octets -> seulement 80 fichiers !

Compilation de busybox

- Récupération des sources sur http://busybox.net
- Configuration de busybox :
  `make menuconfig`
  Choisir un exécutable statique, compilé nativement.
- Compilation de busybox :
  `make`
- Pré-installation de busybox (dans le sous-répertoire `_install/`) :
  `make install`
- Résultat : un exécutable de `500 Ko implémentant toutes les commandes nécessaires` !

Nouvelle compilation de busybox

`500 Ko`, c'est déjà trop important pour un système embarqué parfait !

- Nouvelle configuration de busybox
  `make menuconfig`
  Choisir de construire un exécutable statique “cross-compilé”, en utilisant une chaîne de construction `uClibc` au lieu de la `glibc` standard.
- Compilation de busybox :
  `make`
- Pré-installation de busybox (dans le sous-répertoire `_install/`) :
  `make install`
- Résultat : un exécutable de `250Ko` implémentant toutes les commandes dont nous avons besoin !

Remplir le système de fichiers racine

Connecté en tant que `root` :

- Création d'un point de montage :
  ```
  mkdir /mnt/rootfs
  ```
- Montage de l'image root du système de fichiers :
  ```
  mount -o loop rootfs.img /mnt/rootfs
  ```
- Copie de la structure de fichiers busybox :
  ```
  rsync -a busybox/_install/ /mnt/rootfs/
  chown -R root:root /mnt/rootfs/
  ```
- Synchroniser les modifications dans l'image du système de fichiers monté :
  ```
  sync
  ```

Démarrer le système virtuel

Utilisation de l'émulateur `qemu` comme chargeur de démarrage (pas besoin de copier le noyau dans le stockage cible)

```
qemu \
  -m 32 \
  -hda rootfs.img \
  -kernel linux-2.6.12/arch/i386/boot/bzImage \
  -append "root=/dev/hda clock=pit"
```

Quantité de mémoire (Mo) de la cible émulée

Contenu du disque de la cible émulée

Image du noyau

Ligne de commande du noyau

Création des fichiers périphériques

- Création des fichiers spéciaux des périphériques à chaque fois qu'un programme se plaint :

```
mkdir /mnt/rootfs/dev
mknod /mnt/rootfs/dev/console c 5 1
mknod /mnt/rootfs/dev/null c 1 3
```

- Prendre l'hôte GNU/Linux comme exemple pour trouver les bons numéros de majeur et de mineur :

```
ls -l /dev/console
ls -l /dev/null
```

Monter les systèmes de fichiers virtuels

Rendre disponibles `/proc` et `/sys`
(requis par plusieurs outils en ligne de commande comme `ps`)

- Monter `/proc` :
```
mount -t proc none /proc
```
- Monter `/sys` :
```
mount -t sysfs none /sys
```

Type de système de fichiers

Périphérique brut
ou image du système de fichiers
Dans le cas de systèmes
de fichiers virtuels, tout chaîne convient

Point de montage

Fichier /etc/inittab pour init de busybox

Création du fichier `/etc/inittab` requis par `init` de `busybox`
Récupération d'un exemple provenant de la documentation de `busybox` (mais pas en provenance de l'hôte GNU/Linux... car il dispose de trop de fonctionnalités !)

```
# Ceci est un script pour init
::sysinit:/etc/init.d/rcS
# Démarrer un shell "askfirst" sur la console
::askfirst:-/bin/sh
# Choses à refaire au redémarrage d'init
::restart:/sbin/init
# Choses à refaire au rédémarrage de la machine
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

Configuration du réseau

- Ajout de TCP/IP et du pilote de la carte réseau pour le noyau
- Montage de l'interface réseau :
  `ifconfig eth0 172.20.0.2`
- Utilisation de l'hôte GNU/Linux comme passerelle :
  `route add default gw 172.20.0.1`
- Tests du réseau :
  `ping -c 3 172.20.0.1`
  `-c 3` : utile lorsque `[Ctrl][C]` ne fonctionne pas (configuration tty manquante)
- Tests du routage:
  `ping -c 3 <adresse externe>`

Lancer un serveur HTTP

- Copie des pages HTML dans `/www` (par exemple)
- Création des scripts CGI dans `/www/cgi-bin/`
- Lancement du serveur HTTP de busybox :
  `/usr/sbin/httpd -h /www/ &`

Scrip de démarrage /etc/init.d/rcS

```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
ifconfig eth0 172.20.0.2
route add default gw 172.20.0.1
/usr/sbin/httpd -h /www/ &
/bin/sh
```

Voyez à quel point cela peut être simple !

Un script CGI simplissime

/www/cgi-bin/uptime:

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<html><header></header><body>"
echo "<h1>Durée de fonctionnement:</h1>"
echo "Votre système embarqué tourne
depuis:<pre><font color=Blue>"
echo `uptime`
echo "</font></pre></u>"
echo "</body></html>"
```

Limitations

Quelques limitations mineures

- Scripts CGI : impossible d'implémenter des scripts non triviaux
  Besoin de coder en C pour supporter l'envoi d'informations et l'analyse des URL.
- Logiciel spécifique au système : impossible de faire partie de `busybox`. Il est nécessaire d'ajouter des exécutables C. En conséquence, nécessaire d'inclure la bibliothèque uClibc et de compiler les exécutables avec le support des bibliothèques partagées.

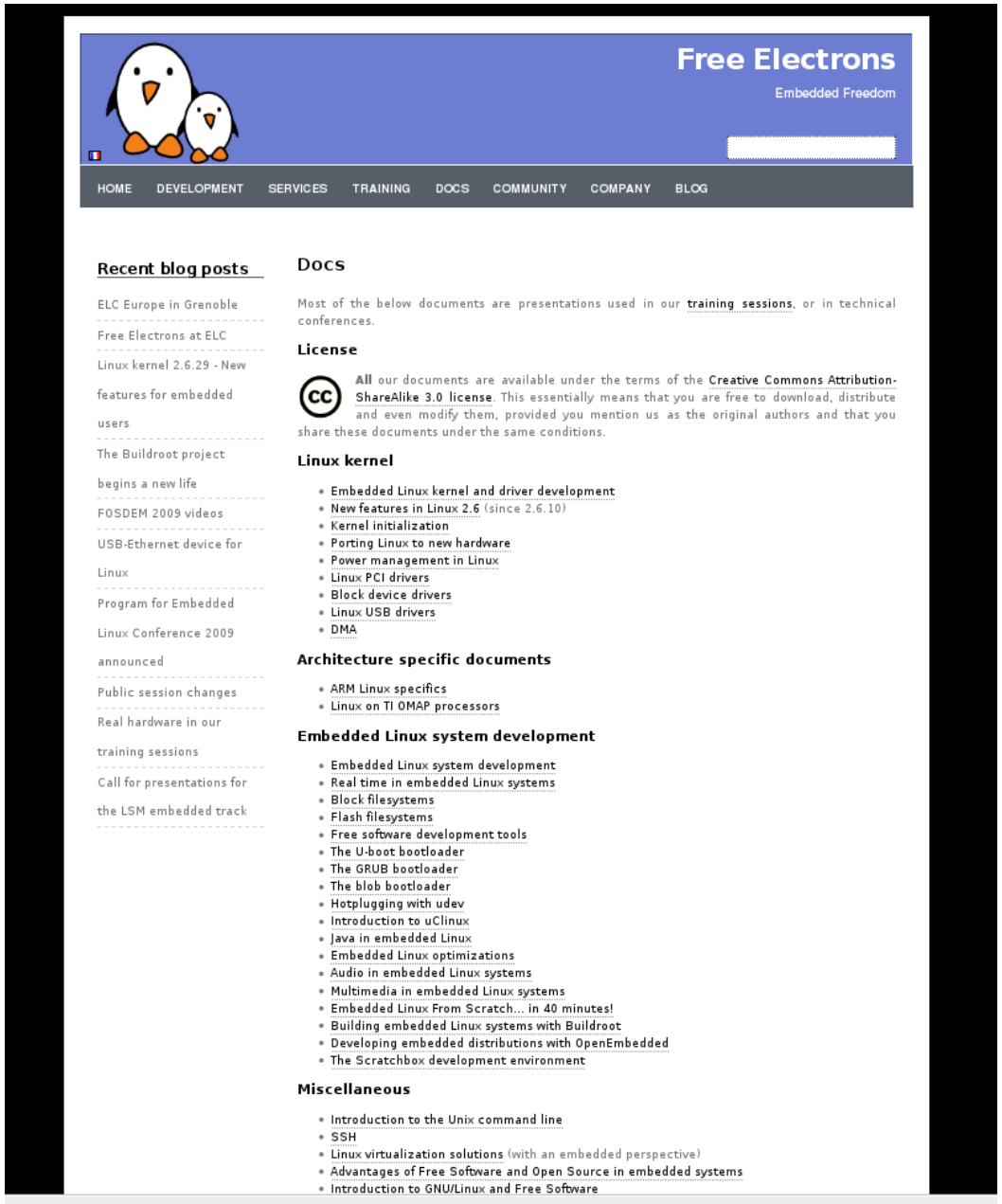
Limitations faciles et peu chères à contourner !

Vrais systèmes embarqués

Ce tutoriel a déjà été effectué sur de vraies cartes de développement !

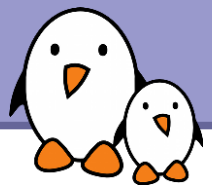
- Nécessaire d'installer et de configurer un chargeur de démarrage (si manquant)
- Nécessaire de transférer les images du noyau et du système de fichiers racine. Une façon efficace d'y arriver est de lancer la cible sur un répertoire NFS exporté à partir de l'hôte GNU/Linux.

Related documents



All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs