EUROGICIEL GROUPE

Formation Linux embarqué

GIFAS

Version du 16 octobre 2010

Sommaire

Linux embarqué ?

Contraintes des systèmes embarqués

- Importance du coût des composants

- Taille RAM et ROM limitées
- Mémoire de masse souvent NOR ou NAND flash
- Systèmes typiques :
  - 4 MB flash, 8 à 16 MB RAM
  - 32 MB flash, 64 à 128 MB RAM

- Différentes architectures de CPU (ARM, MIPS, SH, etc.)
- CPU de faible puissance, souvent sans MMU
- Contrainte de temps : temps réel dur ou mou
- Bus et interfaces différents (I2C, SPI, CAN, etc.)
- Exigence de basse consommation électrique

- Interface homme – machine différente ou inexistante

Contraintes Linux embarqué

- Le CPU doit être supporté par le noyau Linux et la chaine de compilation :
  - CPU 32 bits minimum
  - RAM de 4 MB au strict minimum, 8 MB recommandés
  - ROM de 2 MB au strict minimum, 4 MB recommandés
- Le noyau Linux n'est à l'origine pas architecturé pour être temps réel
  - Le noyau 2.6 n'est préemtable que dans certaines conditions
  - Il existe un patch pour améliorer l'aspect temps-réel :
    - RT-Preempt : http://rt.wiki.kernel.org
  - Il existe des solutions utilisant un micro-noyau exécutant Linux dans un thread :
    - Xenomai : http://www.xenomai.org/
- Le noyau Linux requiert un système de fichier
- Si MMU : mode protégé (séparation espace utilisateur / espace noyau)
- Conflits entre code propriétaire et la licence GPL

Contraintes Linux embarqué (suite)

- Souvent des applications utilisées en Linux embarqué ont été désignées pour un usage sur PC :
  - Les contraintes telles que la quantité de mémoire ou de disque n'ont pas été prises en compte

- Les distributions "classiques" (Fedora, Debian, etc.) utilisent un système de gestion de paquets logiciels inadapté à la génération d'un Linux embarqué
  - Pas de support pour la compilation croisée
  - Pas toujours possible d'installer un paquet dans un répertoire spécifique

Avantages Linux embarqué

- Indépendance d'un vendeur ou éditeur
- Support CPU variés
  - Portabilité
- Time to market
- Faible cout
  - Développement
  - Royalties
- Open source
  - Large communauté
- Suit les standards (POSIX)
  - Portabilité

Distributions commerciales Linux embarqué

- BlueCat Embedded Linux
  - XScale, PowerPC, IA-32, ARM, MIPS, x86
  - http://www.lynuxworks.com
- Cadenux
  - ARM7 et ARM9
  - http://www.cadenux.com
- Denx
  - PowerPC, ARM, MIPS
  - http://www.denx.de
- ELinOS
  - Xscale, PowerPC, x86, ARM, MIPS, SH
  - http://www.sysgo.com
- Metrowerks
  - x86, ARM, PowerPC, ColdFire
  - http://www.metrowerks.com
- MontaVista
  - PowerPC, ARM, MIPS, IA32, SuperH, Xscale, and Xtensa
  - http://www.mvista.com

Outils pour Linux embarqué

FORMATION

EUROGICIEL GROUPE

Outils de compilation

- Le CPU cible est souvent différent de celui de la machine du développeur
  - Il faut une chaine de compilation croisée :
    - Compilateur C / C++, assembleur, éditeur de liens
    - Binutils : ar, Objdump, nm, readelf, etc.
    - Debugger
  - Il faut une librairie C compilée pour la cible

- Il faut parfois compiler la suite gcc, binutils et la librairie C manuellement
  - Long et fastidieux
  - Des outils existent pour simplifier cette tache

Emulateurs

- QEMU
  - Cibles : x86, x64, PowerPC, Sparc et ARM
  - Licence LGPL / GPL
  - http://www.nongnu.org/qemu
- PearPC
  - Cible : PowerPC
  - Licence GPL
  - http://pearpc.sourceforge.net
- Gxemul
  - Cibles : ARM, MIPS, M88K, PowerPC et SuperH
  - Licence BSD
  - http://gxemul.sourceforge.net

Librairies C – GNU Glibc

- Une libraire C est nécessaire à tout système Linux
- La libraire standard GNU C (Glibc) est la plus complète, la plus utilisée, respectant le mieux les standards
  - Support de l'internationalisation
  - Support des threads Posix natifs (NPTL)
  - Support de la STL C++
- Licence LGPL
- Assure une bonne compatibilité avec des composants ou librairies tiers
- Souvent de taille trop importante pour l'embarqué
  - Entre 1.5 et 2 MB
- Il existe des alternatives à Glibc pour les systèmes embarqués
  - Implémentation d'une librairie C de petite taille
  - Fournissant la plupart des fonctionnalités de Glibc

Librairies C – uClibc

- Librairie associée au projet uClinux
  - uClibc est dérivée d'anciennes librairies du projet
- Supporte les processeurs avec ou sans MMU
- Vise à maintenir la compatibilité avec C89, C99, and SUSv3
- Support de la STL disponible avec uClibc++
- Environ 4 fois plus légère que Glibc
  - ~400 KB sur ARM
- Licence LGPL
- Fournit un environnement de compilation : BuildRoot
- http://uclibc.org

Librairies C – Eglibc

- Projet récent ayant pour but d'optimiser la Glibc
  - Compatibilité d'API et d'ABI avec Glibc
- Configuration des composants inclus à la compilation
- Licence LGPL
- http://www.eglibc.org

Librairies C – Dietlibc

- Licence GPL ou commerciale
- dietlibc a été écrit "from scratch" avec pour but de minimiser la taille tout en augmentant les performances
- Désignée pour être liée statiquement (mais peut être chargée dynamiquement)
- Supporte Alpha, ARM, ia64, MIPS, s390, Sparc, x86 et PowerPC
- Très légère :
  - Environ 70 KB
- http://www.dietlibc.org/
- http://www.fefe.de/dietlibc/

Environnements – GCC

- Une chaine de compilation croisée est nécessaire pour compiler sur une machine hôte d'architecture différente de la cible

- GCC est le compilateur standard sous Linux
  - Difficile de compiler le noyau avec un autre compilateur
  - Il existe des patches pour Intel CC

- Recompiler GCC pour générer une chaine croisée n'est pas trivial
  - http://gcc.gnu.org/install/

- Il existe des outils, simples scripts ou environnements complets, permettant de compiler GCC, les outils (binutils) et une librairie C

Environnements – Crosstool-NG

- Crosstool-NG est un ensemble de scripts pour générer une chaine de compilation croisés
- Supporte les architectures Alpha, ARM, MIPS, PowerPC, SH, s390 et x86
  - 64 bits support souvent à l'état expérimental
- Supporte Glibc, uClibc, Eglibc
- Outil de configuration ncurses type menuconfig
- http://ymorin.is-a-geek.org/projects/crosstool

Environnements – ELDK

- Embedded Linux Development Kit
- Targets : PowerPC, ARM et MIPS
- Contient une chaine de compilation croisée, debugger, des librairies précompilées
  - Code source, patches, scripts, etc. sont disponibles gratuitement
- Supporte Glibc et uClibc
- Paquets et installation basés sur RPM
- Support commercial disponible
- http://www.denx.de/wiki/DULG/ELDK

Environnements – Buildroot

- Buildroot est un outil permettant de générer tout un Linux embarqué, de la chaine de compilation jusqu'au root FS
  - Composé de Makefile et de scripts
  - A l'origine, outil de test pour uClibc

- Buildroot automatise les séquences :
  - Téléchargement des sources
  - Configurations (à travers une interface ncurses type menuconfig)
  - Compilation de la chaine de compilation croisée (si basée sur uClibc)
  - Compilation des paquets
  - Création du système de fichier

- Gère les dépendances entre paquets

- http://buildroot.net/

Environnements – Bitbake

- Bitbake est un outil similaire à Buidroot
  - Dérivé de l'outil Portage de la distribution Gentoo (langage Python)
  - Fondation du projet OpenEmbedded
    - http://www.openembedded.org/
- Supporte d'autres cibles que Linux : les BSDs, Cygwin, etc.
- Des recettes gèrent les dépendances, le téléchargement, la configuration, la compilation, l'installation et la désinstallation d'un paquet
- Les recettes incluent des métadonnées sur le paquet, qui peuvent être conditionnelles, et être héritées entre paquets
  - Permet de prendre en compte un héritage de configuration
- La configuration se fait par édition de fichiers
- http://bitbake.berlios.de/
- http://wiki.openembedded.net/

Interfaces graphiques

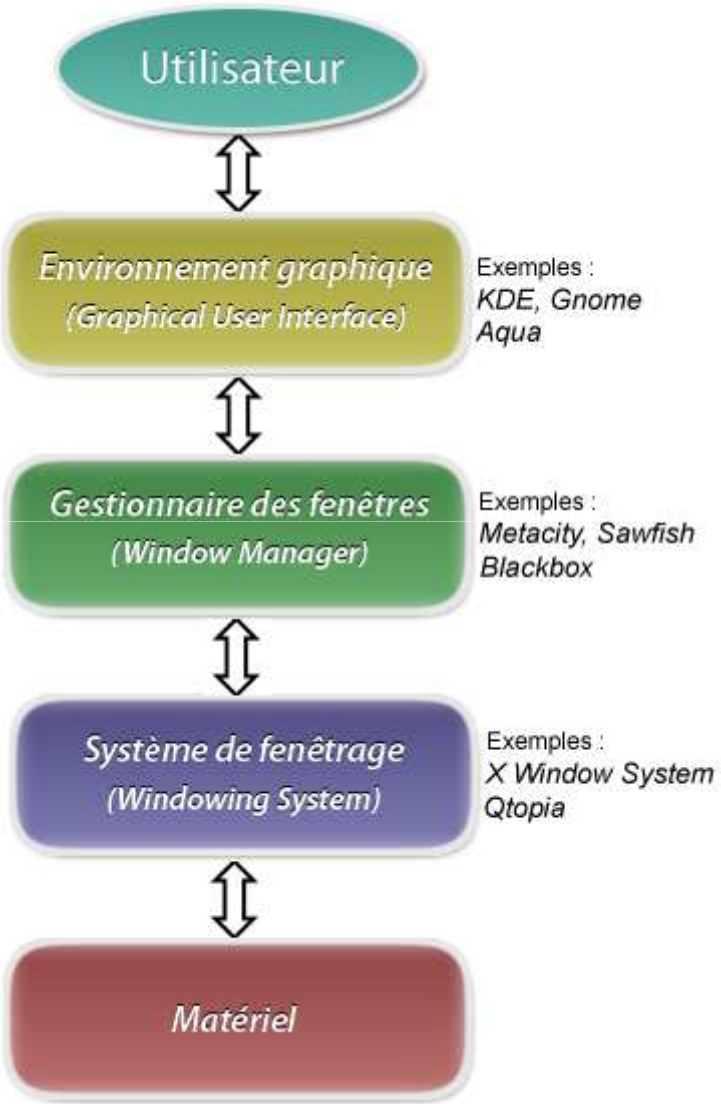
FORMATION

EUROGICIEL GROUPE

IHM – Introduction

- Sur un desktop Linux, l'interface graphique est basée sur le système de fenêtrage X Windows System (ou X11), d'un gestionnaire de fenêtres et d'un environnement graphique

- Pas adapté aux contraintes de l'embarqué :
  - X11 : 5 MB ram, 16 MB disque
  - Gnome : 14 MB ram, 96 MB disque
  - KDE : 11 MB ram, 26 MB disque



IHM – Framebuffer Linux

- Il existe un certain nombre de solutions dédiées à Linux embarqué, le plus souvent utilisant le framebuffer virtuel du noyau
- Il ne repose pas sur des librairies spécifiques comme SVGALib ou lourdes comme X11
- Le framebuffer Linux (fbdev) présente une couche abstraite du hardware, garantissant une carte mémoire fixe quelque soit le hardware graphique utilisé
- Il permet donc aux applications d'accéder au hardware graphique à travers une API bien définie
  - ioctl sur /dev/fb0
  - Préférable d'utiliser une librairie intermédiaire comme DirectFB ou SDL

IHM – Interfaces framebuffer et toolkits

DirectFB www.directfb.org	LGPL	1 à 1.5 MB ROM
Nano-X www.microwindows.org	GPL MPL	<100 KB ROM, 50 à 250 KB RAM
KDrive / Tiny-X www.x.org	X11	5.4 MB ROM
Qt qt.nokia.com	LGPL Commerciale	16 MB ROM
GTK/DFB www.gtk.org	LGPL	4.5 MB ROM
FLNX www.fltk.org	LGPL	
MiniGUI www.minigui.com	GPL	1 à 3 MB ROM Jusqu'à 8 MB RAM
WxEmbedded www.wxwidgets.org	LGPL	1 à 2.5 MB ROM

IHM – DirectFB

- DirectFB est une librairie légère qui supporte les accélérateurs graphiques hardware
- Abstraction des périphériques d'entrée
- Infrastructure de système de fenêtrage, supportant les fenêtres transparentes
- Par défaut, n'exécute qu'une seule application
  - Nécessite une extension pour exécuter plusieurs applications
    - Fusion ou SaWMan
- Souvent utilisée comme couche bas-niveau pour d'autres librairies graphiques
- Très populaire et bien documentée

IHM – DirectFB – Exemple

FORMATION

EUROGICIEL GROUPE

IHM – Nano-X

- Licence MPL ou GPL
  - La licence Mozilla Public License autorise la création de drivers et applications propriétaires, mais le code de Nano-X lui-même doit rester ouvert
- Développé spécialement pour les systèmes embarqués
- Supporte différents types d'écrans et de périphériques d'entrée (souris, touch screens, claviers)
- Implémente deux APIs
  - Microwindows : API Win32 permettant le portage d'application Windows ou Windows CE
  - Nano-X : API similaire à X-lib
- Pas de nouvelle release depuis 2005

IHM – Nano-X – Exemples

- Exemple de GUI: http://www.microwindows.org/ScreenShots.html

PIXIL PDA v1.1

Webmedia Linux4.TV sur Nano-X server

IHM – MiniGUI

- Licence GPL ou commerciale
  - La version GPL est limitée en fonctionnalités
  - Nécessite une licence commerciale pour les applications propriétaires
- Supporte les OS embarqués les plus courants : Linux, eCos, uC/OS-II, VxWorks, pSOS, ThreadX et Nucleus
- Trois mode d'exécution : threads, processus et single-task
- Populaire en Asie

IHM – MiniGUI

- Exemple d'application: http://www.minigui.org/whatis-app.shtml

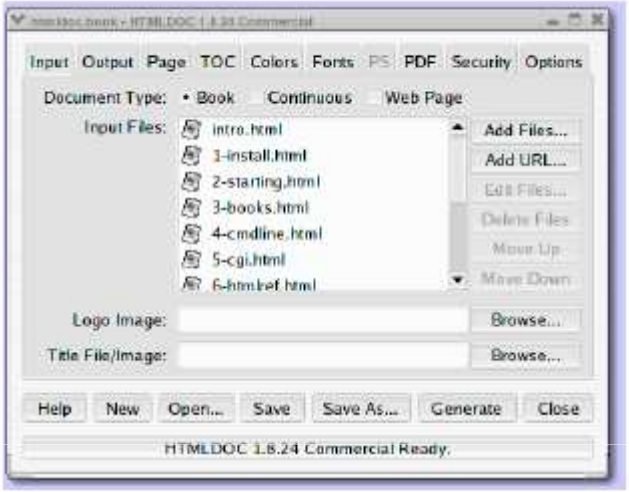
Browser web sur STB

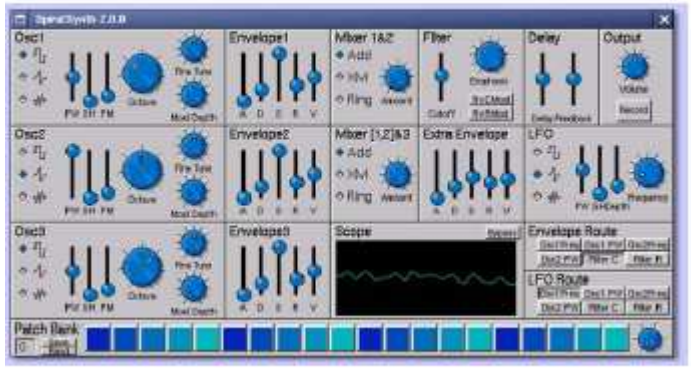
Open source PMP (Portable Multimédia Player)

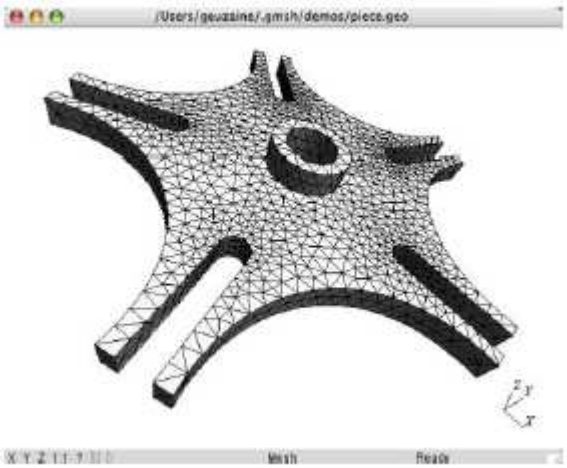
IHM – FLNX et FLTK

- FLTK supporte X11, Windows et MacOS X
- FLNX est un portage de FLTK sur Nano-X (voir site Nano-X)
- C++
- Supporte graphiques 3D avec OpenGL
- Environnement de conception graphique (FLUID builder) ; permet de prototyper une application en peu de temps

IHM – FLTK – Exemples







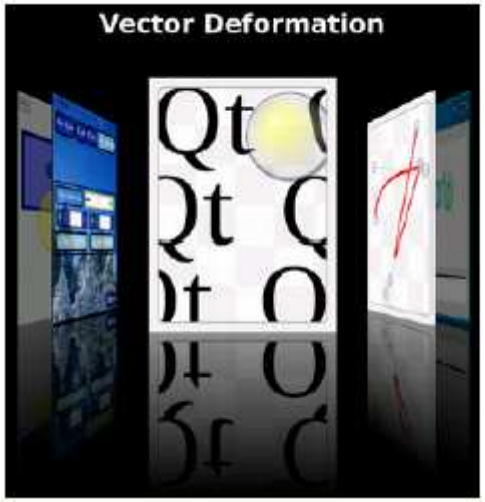
FORMATION

EUROGICIEL GROUPE

IHM – QT

- Deux versions :
  - Licence LGPL
  - Licence commerciale
- C++
- QT est le toolkit de KDE
- Architecture client / serveur
- Basé sur le framebuffer, X11 ou DirectFB
- Intégré par plusieurs Java VM
- Outils de développement

IHM – QT – Exemples



IHM – KDrive

- Auparavant appelé Tiny-X
- Version simplifiée du serveur X11 pour systèmes embarqués
- Supporte entièrement le protocole X11
- Permet d'utiliser n'importe quelle application ou librairie X11
- Permet de programmer une application sur l'API X-lib
- Est normalement utilisé avec un toolkit tel que Gtk ou Fltk

IHM – Gtk/DFB

- Gtk est le toolkit utilisé pour les applications Gnome
  - API standardisée
- Gtk/DFB est une version pour DirectFB
- Gère par défaut une seule application
  - Voir Matchbox, un gestionnaire de fenêtre simple et léger
- Graphiques vectoriels basé sur la librairie Cairo
- Exemple : www.directfb.org/docs/GTK_Embedded/

IHM – WxEmbedded

- A l'origine, WxWidget est une librairie d'abstraction au dessus de différents toolkits (Win32, Gtk, DirectFB, etc.)
- WxUniversal et WxEmbedded ont étés développés pour permettre de développer des applications WxWidget sans utiliser de toolkit
- Supporte X11, DirectFB et Nano-X
- C++

IHM – Conclusions

- Il y a un large choix de solutions pour développer une interface graphique embarquée :
  - Composants de bas niveau :
    - DirectFB, Tiny-X, Nano-x
  - Composants de haut niveau :
    - GTK, QT, FLTK, WxEmbedded, miniGUI

- De part leurs licences, leur maturité, leur vitalité, les projets suivant sont les plus souvent utilisés :
  - DirectFB, Tiny-X, GTK et QT

Systèmes de fichiers

Séquence de boot

- Le démarrage d'un système Linux peut être résumé à 3 étapes :
  - Boot loader
    - Initialise le hardware
    - Charge le noyau
    - Transfère le contrôle au noyau
  - Initialisation du noyau
    - Initialisation et démarrage des sous-systèmes du noyau
    - Démarrage du multitâches
    - Montage du système de fichier racine
    - Passe le contrôle au mode user
  - Initialisation en espace user
    - Démarrage des services et applications

Séquence de boot – Boot loader

- Une fois que le boot loader a initialisé le hardware, il doit démarrer le noyau :
  - Chargement du noyau (depuis une mémoire flash ou le réseau)
  - Décompression du noyau
  - Si un disque mémoire initial (initial ram disk = initrd) est requis, il est chargé en mémoire par le boot loader
  - Initialisation d'une zone mémoire pour passage de paramètres au noyau (avec éventuellement l'adresse de l'image initrd)
  - Appelle le point d'entrée du noyau
  - Généralement le noyau pourra récupérer l'espace mémoire du boot loader qui n'est plus utilisé

Séquence de boot – Noyau

- Le démarrage du noyau se fait en plusieurs phases :
  - Initialisation du hardware (CPU, MMU, caches, stack, etc.)
  - Analyse des paramètres passés par le boot loader
  - Si un initrd est utilisé, il est monté comme système de fichiers racine
  - Autres initialisations (mémoire virtuelle, interruptions, timers, etc.)
  - Initialisation des drivers
  - Exécute `/init` si présent dans l'initrd
  - Montage du système de fichiers "maitre" (disque, flash, NFS, etc.), éventuellement en place du système initrd
  - Exécute en espace user le processus init

Séquence de boot – Espace User

- Le processus init est exécuté en espace user par le noyau
  - Généralement `/sbin/init`

- Le processus init est spécial :
  - Il ne peut jamais être terminé
  - Il adopte les processus orphelins (processus fils dont le parent est terminé en premier)
  - Le noyau informe le processus init de certains événements comme Ctrl-Alt-Del

- Le processus init a pour tache de lancer les autres services et applications du système
  - Généralement, ces actions sont spécifiées dans `/etc/inittab`

Filesystems – Ramfs et Tmpfs

- Ramfs est un système de fichiers, simulé en mémoire par le noyau (dans le cache dentry)
  - Pas de limite de taille
  - Peut exhausser toute la mémoire

- Tmpfs est dérivé de Ramfs (utilise des caches internes au noyau), mais :
  - La taille du système varie dynamiquement en fonction des fichiers contenus
  - Une limite de taille maximum est spécifié au montage
    - Peut être changée "on-the-fly" avec `mount`
  - Tmpfs peut utiliser le swap (pas Ramfs)

- Un système Linux utilise souvent un répertoire comme `/tmp` pour stocker des informations temporairement (qui n'ont pas besoin de persister un reboot)
  - Il est possible d'utiliser un système en mémoire Tmpfs pour `/tmp`
    - `/var/run`, `/var/lock`, etc peuvent être des symlinks sur `/tmp`
  - Conseillé si le système est monté sur une mémoire flash

Filesystems – Initrd

- Lors du boot, le noyau doit être en mesure de monter le système de fichiers et d'exécuter le processus `init`
- Si pour accéder au périphérique contenant le système de fichier, le noyau doit charger un driver compilé en module, on a un problème de l'œuf et de la poule :
  - Le système de fichiers contient le module
  - Le module est nécessaire pour accéder au système de fichiers
- Une solution consistait à utiliser un disque RAM qui contenait les modules nécessaires
- Ce disque RAM était chargé depuis un fichier par le boot loader puis monté par le noyau
  - Le noyau pouvait alors charger les modules nécessaires au montage du système de fichiers normal
- Ce mécanisme est remplacé par initramfs depuis 2.6 (bien que toujours possible)

Filesystems – Initramfs

- Tout noyau 2.6 contient une archive cpio compressée (gzip)

- Au démarrage :
  - L'archive est extraite en mémoire (tmpfs)
  - Monté comme rootfs
  - Si le fichier `/init` est présent
    - Le noyau l'exécute avec le PID 1
    - `/init` ne retourne jamais
  - Sinon, le noyau montera une partition disque et exécutera `/sbin/init`

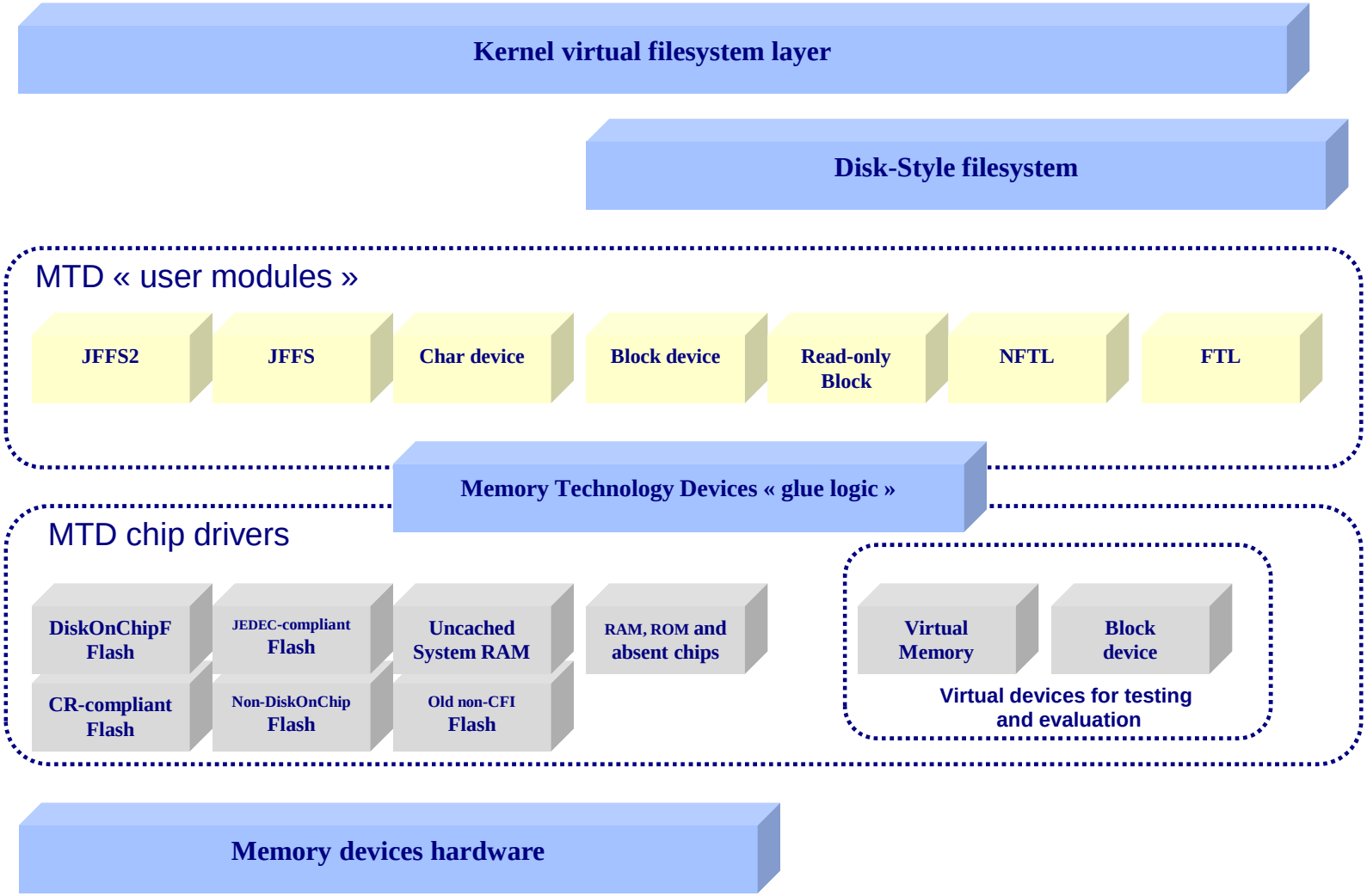
- Initramfs est LE rootfs, le processus `init` se chargera de monter un autre système à sa place

- Comparé à initrd, initramfs est plus simple, flexible et efficace en terme de mémoire
- Initramfs peut être utilisé comme le seul système racine du système (plus besoin de compiler un seul driver de système de fichiers ou de périphérique type bloc)

Filesystems – MTD

- Memory Technology Devices incluent les mémoires telles que :
  - ROM (PROM, EPROM)
  - NOR flash
  - NAND flash
- Le noyau dispose d'un sous système MTD présentant une abstraction uniforme quelques soient les technologies mémoires utilisées
- Toutes les technologies mémoires ne sont pas gérées par MTD (memory stick, Compact-Flash, Secure Digital, USB drives, etc.)
- http://www.linux-mtd.infradead.org

Filesystems – MTD (suite)



Filesystems – Cramfs

- Cramfs = Compressed RAM File System

- Système de fichier à lecture seule, compressé
- Peut être utilisé avec :
  - NOR et NAND flash (via MTD ou FTL/NFTL)
  - Compact-Flash (IDE)
  - Disque RAM

- La taille maximale du système est de 272 MB, celle d'un fichier est de 16 MB
- Il n'y a pas de répertoire courant (.) ou parent (..)
- Les dates des fichiers sont Epoch (1 janvier 1970, 00:00:00 GMT)
- Les UIDs ont une taille de 16 bits et les GIDs de 8 bits (normalement 32 et 16 respectivement)
- La compression utilisée est zlib, faite par blocs de 4 KB

- L'image chargée en flash est générée avec `mkcramfs` (avec la même endianess que le système hôte)

Filesystems – Squashfs

- Squashfs est similaire à Cramfs, mais avec des limitations moindres

- Système de fichier à lecture seule, compressé

- Il supporte les UIDs et GIDs de 32 bits et les dates de création des fichiers
- Les tailles maximales du système de fichier et d'un fichier sont de 2^{64} octets
- Les tailles des blocs peut aller jusqu'à 1 MB (64 KB par défaut)
- La compression utilisée est gzip (un patch pour lzma existe)
- Supporte big et little endian

- L'image est créée avec mksquashfs

- Les fichiers dupliqués sont détectés et effacés
- Intégré au noyau depuis 2.6.29

Filesystems – JFFS2

- Journaling Flash File System version 2
- JFFS2 est spécifiquement désigné pour les flash NOR et NAND (via MTD)
- Compressé (zlib, rubin et rtime)
- Réparti l'usure sur la mémoire flash et empêche les effacements d'être trop concentrés
- Les changements sur les fichiers et répertoires sont enregistrés dans la flash dans des nœuds
  - Basé sur une concept de blocs (de la taille des segments d'effacement de la flash) propres ou sales, et d'un garbage collector
- L'arborescence des inodes n'est pas enregistrée dans le système
  - Tous les nœuds doivent être examinés au moment du montage
  - Potentiellement lent selon la taille de la flash

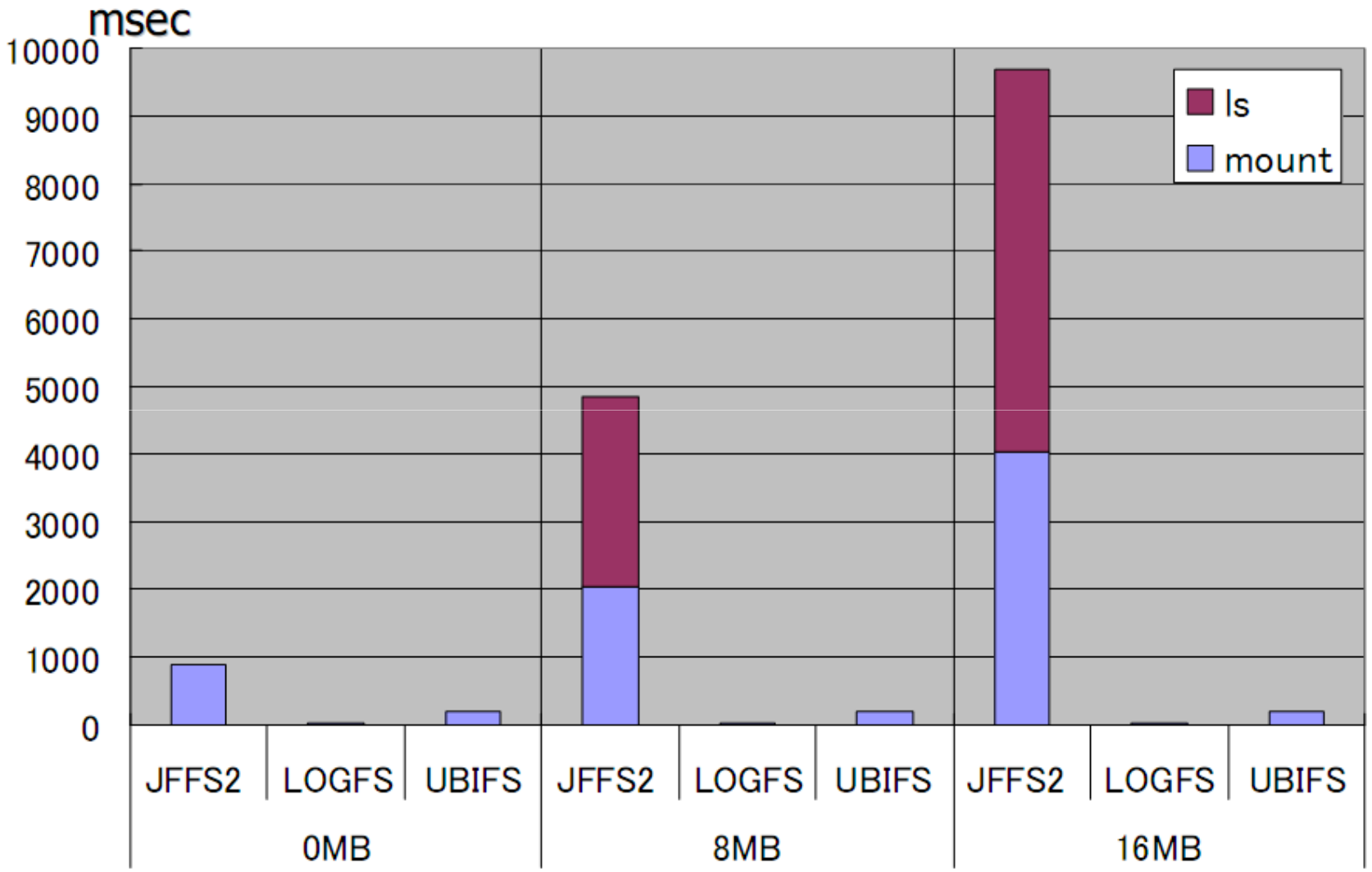
Filesystems – LogFS

- Développé comme remplaçant de JFFS2, pour les flash de grande taille
- Contrairement à JFFS2, l'arborescence des inodes est enregistrée dans le système
  - Temps de montage extrêmement rapides
  - Un bloc modifié doit être déplacé
    - Ecritures potentiellement plus lentes que JFFS2
  - Wear leveling
- Intégré au noyau depuis 2.6.34

Filesystems – UBI / UBIFS

- Unsorted Block Images (UBI) est un système de gestion de volumes (via MTD)
  - Wear leveling
- UBIFS est un système de fichiers utilisant des volumes UBI
- Développé comme remplaçant de JFFS2 (compétiteur de LogFS)
- Beaucoup plus rapide que JFFS2 (write caching)
- Le temps de montage du système de fichier est indépendant de la taille (contrairement à JFFS2)
- Intégré dans le noyau depuis 2.6.27

Filesystems – Benchmark



Filesystems – Ext2

- Ext2 (Second Extended Filesystem) est l'un des premiers systèmes de fichiers de Linux
- Conçu pour les périphériques type bloc :
  - Disque dur
  - Disque RAM
  - Compact-Flash (IDE), SD, SSD, USB flash drives
  - Flash NAND ou NOR en mode lecture seule (via MTD ou FTL/NTFL)
- Très rapide
- Pas de mécanisme de récupération des données (journal)
- Tailles maximales : système : 16 TB, fichier : 2 TB

Filesystems – Ext3 et Ext4

- Ext3 est une extension de ext2 ajoutant un journal
- Mécanisme de récupération des données par journal
- Compatible avec ext2 :
  - La première fois qu'un système ext2 est monté en tant qu'ext3, le driver ajoute automatiquement un journal
  - Un système ext2 peut être converti en ext3 avec mke2fs
- Tailles maximales : système : 16 TB, fichier : 2 TB
- Ext4 est le successeur d'ext3
- Ext4 a été accepté dans le noyau 2.6.28
- Plus rapide qu'ext3 (delayed allocation)
- Compatible avec ext2 et ext3
- Tailles maximales : système : 1 EB, fichier : 16 TB

Filesystems – XFS

- XFS est un système de fichiers journalisé créé par Silicon Graphics en 1993

- Adressage 64 bits, particulièrement adapté pour les fichiers de grande taille
- Recouvrement du système après interruption très rapide
- Variable size extents

- Taille maximale du système de fichiers : 8 EB (16 TB sur Linux 32 bits)
- Le journal peut-être placé sur un disque différent du système de fichier

- Sous licence GPL depuis 2000, porté sur Linux 2.4 en 2001

Filesystems – Swap

- Sur un système avec MMU, lorsqu'il n'y a plus de pages libres en mémoire, une page sera transférée sur une mémoire de masse (typiquement disque dur), appelée zone d'échange ou swap

- Le système de swap peut être désactivé dans la configuration de build du noyau
  - Attention aux fuites mémoire !

- La décision d'utiliser une partition de swap dépend de :
  - Le type de mémoire de masse utilisée
    - Eviter un swap sur mémoire flash, Compact-Flash, etc
    - Pas de problème avec un disque dur

  - Le ratio quantité de RAM / frugalité des applications
    - Tant qu'il y a de la RAM disponible, le swap n'est pas utilisé

Filesystems – Que choisir ?

- Considérations à prendre en compte lors du choix de systèmes de fichiers pour une application embarquée :
  - Ecriture
    - Est-il nécessaire de pouvoir ajouter, effacer ou modifier des fichiers ?
  - Persistance
    - Le contenu du système de fichiers doit-il être préservé après un shutdown ou redémarrage ?
  - Résilience
    - Doit-on pouvoir récupérer les données en cas d'arrêt brutal (coupure d'alimentation, crash) alors que le système de fichiers n'a pas été synchronisé ni démonté ?
  - Compression
  - Répartition de l'usure (wear leveling)

Filesystems – Exemple

- L'exemple suivant montre l'organisation possible d'une mémoire flash de 4 MB

Raw partition for boot loader	256 K
Raw partition for kernel	640 K
CRAMFS partition for RO data	2 M
JFFS 2 partition for RW data	1.2 M

Filesystems – NFS

- Network File System est un protocole qui permet à une machine d'accéder à des fichiers à travers un réseau
- Principalement utilisé dans le monde Unix, même si supporté sur Windows ou Mac OS
- Compatible avec IPv6
- NVS versions 1 et 2 utilisent UDP et sont non sécurisées
- NFS version 3 utilise UDP ou TCP, avec sécurité élémentaire
- NFS version 4 a été entièrement repensée (nouveau protocole) avec gestion totale de la sécurité, reprise sur incidents, etc.
- Le noyau Linux peut monter un système de fichiers racine distant via NFS

Filesystems – Bootp / DHCP / TFTP

- Bootp (Bootstrap Protocol) sert à une machine à obtenir une adresse IP d'un serveur
- Bootp est aussi utilisé, pour des machines diskless, à obtenir l'adresse IP d'un serveur contenant une image boot (noyau)
- DHCP (Dynamic Host Control Protocol) sert aussi à donner une configuration réseau à une machine, mais est plus récent que Bootp. La plupart des serveurs
- DHCP intègrent les fonctionnalités de Bootp
- Trivial File Transfer Protocol (TFTP) est une forme très basique de FTP
- Il est utilisé par une machine diskless pour récupérer une image boot

Filesystems – Network boot

- Durant le développement d'un système embarqué, il est avantageux de pouvoir télécharger un noyau avec TFTP et/ou de monter un système de fichiers via NFS
- Le système embarqué doit donc avoir une interface réseau
- Pour télécharger un noyau, il faut un boot loader qui supporte TFTP et éventuellement Bootp/DHCP
- Le système de fichiers racine peut être monter à travers NFS :
  - Pas besoin de recharger une image flash à chaque modification
  - Pas de contrainte de taille, les binaires peuvent être compilés avec les informations de débogage

Filesystems – Network boot – Serveur DHCP

- Exemple de configuration d'un serveur DHCP (`/etc/dhcpd.conf`)

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    option routers 192.168.1.254;
    option subnet-mask 255.255.255.0;
    host target {
        hardware ethernet 00:11:e0:02:df:4d;
        fixed-address 192.168.1.201;
        option host-name "target";
        next-server 192.168.1.3;
        filename "/data/kernel/vmlinuz-2.6.24.img";
        option root-path "/data/rootfs/";
    }
}
```

Filesystems – Network boot – Serveur TFTP

- Typiquement sous Linux, le serveur TFTP est géré par inetd ou xinetd
- Exemple de configuration de xinetd (`/etc/xinetd.d/tftp`) :

```
service tftp
{
        socket_type = dgram
        protocol    = udp
        wait        = yes
        user        = root
        server      = /usr/sbin/in.tftpd
        server_args = /data/kernel
        disable     = no
        per_source  = 11
        cps         = 100 2
}
```

FORMATION

Filesystems – Network boot – NFS

- Exemple de configuration d'un serveur NFS
  - Ajouter dans le fichier `/etc/exports` :

    ```
    /data/rootfs 192.168.1.201(rw,no_root_squash)
    ```

    - L'option `no_root_squash` est importante ; elle permet au client d'accéder au répertoire partagé avec les privilèges root
  - Eventuellement éditer les fichiers `/etc/hosts.deny` et `/etc/hosts.allow`

    ```
    portmap: 192.168.1.201
    lockd: 192.168.1.201
    rquotad: 192.168.1.201
    mountd: 192.168.1.201
    statd: 192.168.1.201
    ```

  - Eventuellement éditer la configuration du pare-feu

Filesystems – Network boot – NFS

- Démarrage du serveur NFS

```
$ /etc/init.d/nfs-kernerl-server start
```

- Le noyau de la cible doit être compilé avec le support client NFS
  - `CONFIG_NFS_FS=y`
  - `CONFIG_IP_PNP=y`
  - `CONFIG_ROOT_NFS=y`

- Passer les paramètres au noyau :
  - `root=/dev/nfs ip=192.168.1.201`
    `nfsroot= 192.168.1.3:/data/rootfs`

- Configuration d'un serveur NFS sur Fedora :
  http://optics.csufresno.edu/~kriehn/fedora/fedora_files/f9/howto/nfs.html

Filesystems – Bootloaders

- Dans les systèmes embarqués, la séquence de boot de bas-niveau est dépendante du processeur et de l'architecture hardware
- Le processeur commence à exécuter du code en flash NOR
  - Le boot loader doit être en flash, à la bonne adresse
- Certains processeurs intègrent un bootcode en ROM chargeant un portion de flash en ram
  - Un premier petit boot loader (first stage) charge le boot loader principal
- Le boot loader démarre juste après le reset du processeur et doit donc faire certaines initialisations (ex. contrôleur DRAM)

Filesystems – Principaux bootloaders

	x86	ARM	PPC	MIPS	SH
GRUB www.gnu.org/software/grub/	X				
Syslinux syslinux.zytor.com	X				
Blob www.sf.net/projects/blob/		X			
Bootldr www.handhelds.org/sources.html		X			
uMon microcross.com/html/micromonitor.html	X	X	X	X	X
Redboot sources.redhat.com/redboot/	X	X	X	X	X
U-Boot www.denx.de/wiki/U-Boot	X	X	X	X	X

Filesystems – U-Boot

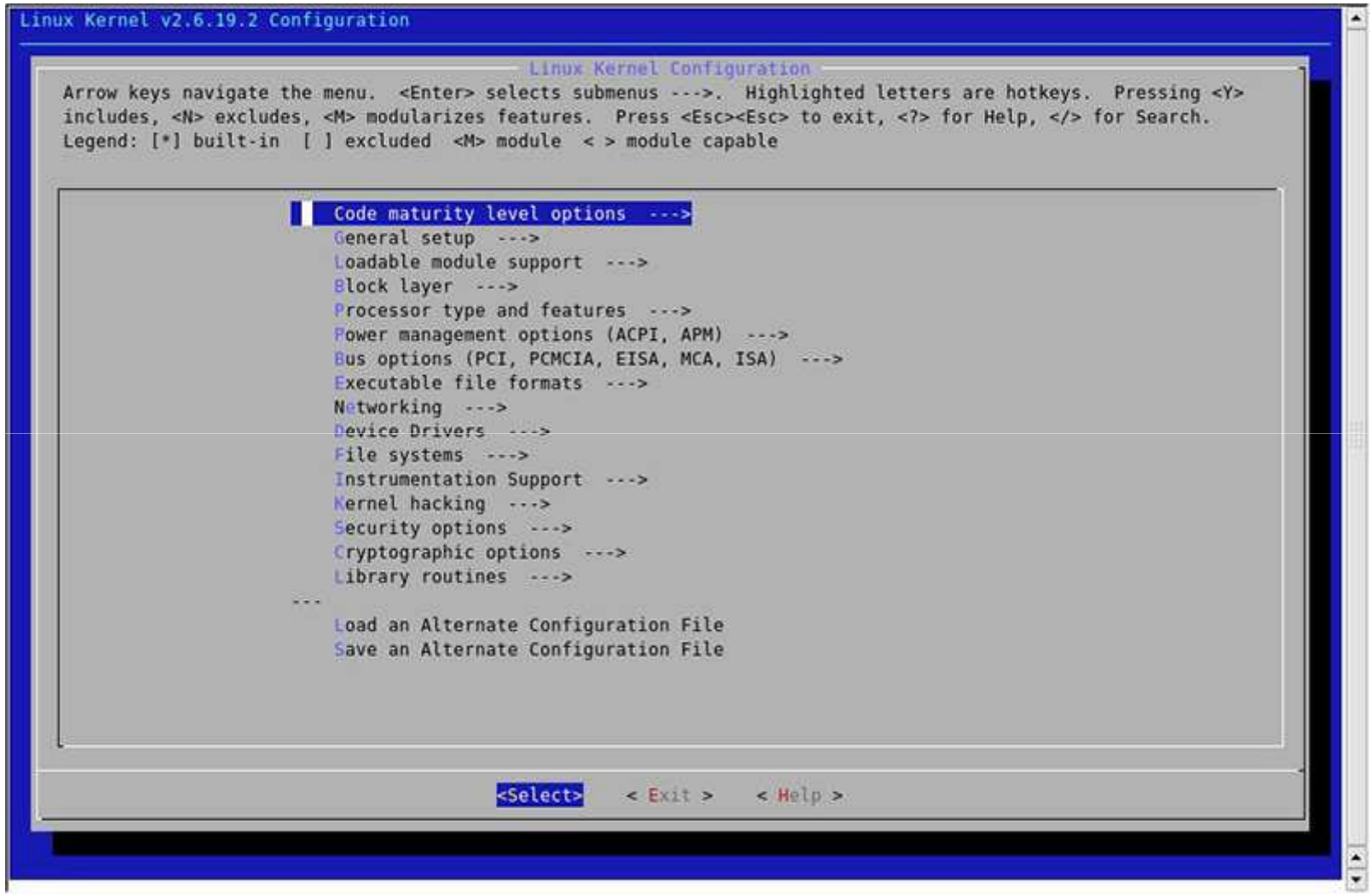
- Das U-Boot (Universal Bootloader ou Das Unterseeboot) est certainement le bootloader le plus flexible et l'un des plus utilisés
- Activement développé, bien documenté
- Capable de booter un noyau en flash, sur disque IDE ou SCSI, depuis USB, et par TFTP
- Supporte Cramfs, ext2, FAT, JFFS2
- Supporte une console sur RS-232
- Permet de charger le noyau et le système de fichier en flash
- Comporte des outils de diagnostiques (lecture / écriture de flash, test de périphériques)

Configuration du noyau Linux

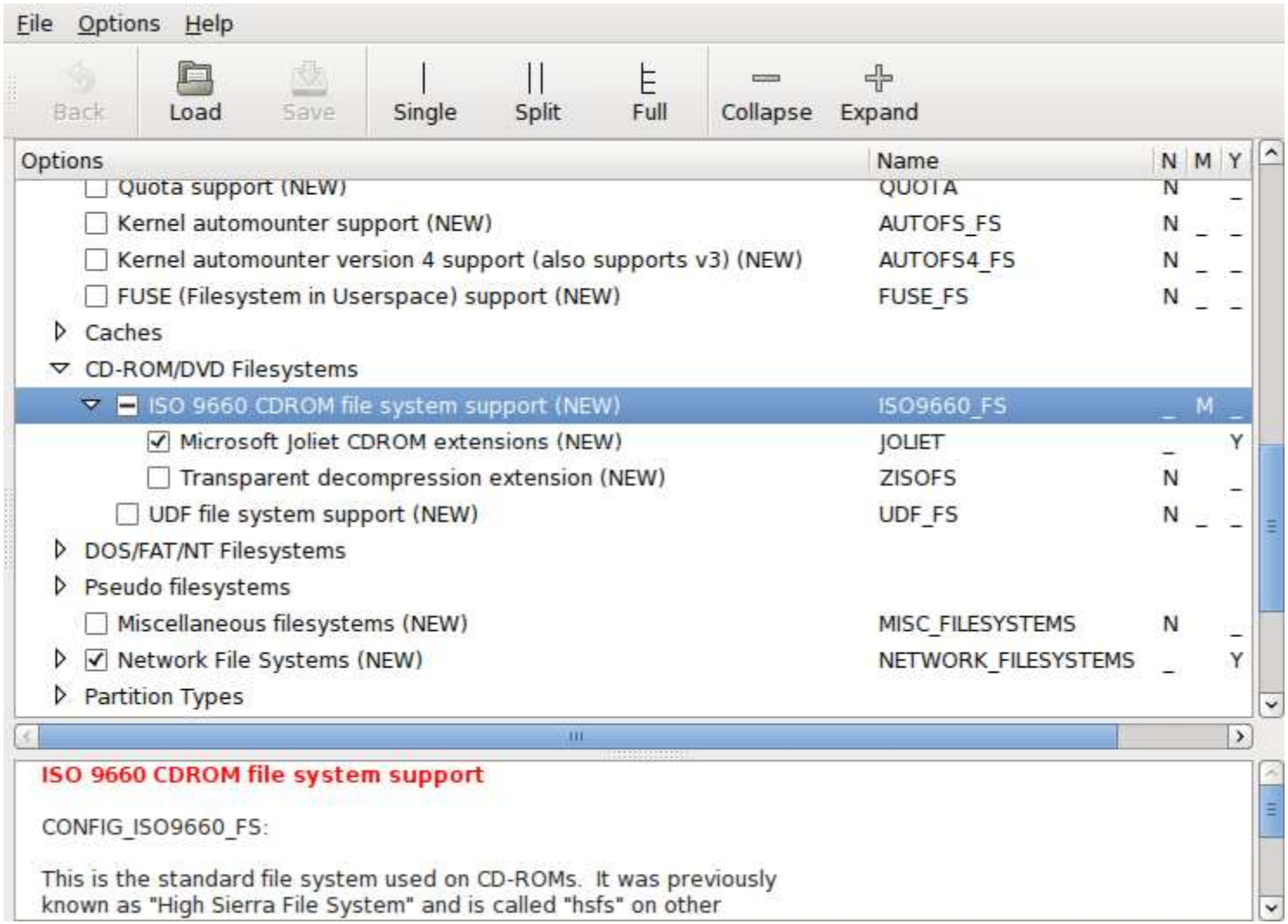
Noyau Linux – Configuration

- Sources officielles du noyau Linux : http://www.kernel.org
- La configuration du noyau est sauvée dans le fichier `.config` (situé à la racine des sources)
- L'outil de configuration a plusieurs interfaces :
  - `make menuconfig`
  - `make xconfig`
  - `make gconfig`
- `xconfig` et `gconfig` sauve l'ancienne configuration dans `.config.old`
- La configuration d'un noyau s'exécutant peut être récupérée (si cette option avait été configurée) :
  - `$ zcat /proc/config.gz`

Noyau Linux – Configuration (suite)



Noyau Linux – Configuration (suite)

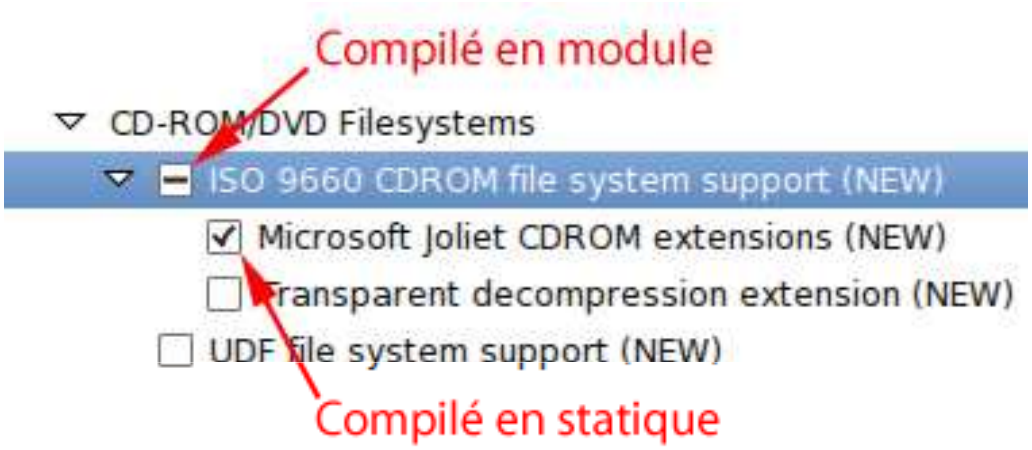


FORMATION

EUROGICIEL GROUPE

Noyau Linux – Configuration (suite)

- Options de configuration :



- Partie du fichier .config correspondante :

```
#
# CD-ROM/DVD Filesystems
#
CONFIG_ISO9660_FS=m
CONFIG_JOLIET=y
# CONFIG_ZISOFS is not set
```

Noyau Linux – Configuration

- Pour utiliser le fichier .config d'une version de noyau à l'autre, il faut le mettre à niveau avec :
  - `make oldconfig`
  - Recommandé aussi si édition manuelle du `.config`
- Pour ne sélectionner qu'une configuration minimale :
  - `make allnoconfig`
- Pour configurer le noyau pour une architecture différente :
  - `make ARCH=arm`
  - `make ARCH=arm defconfig`
  - `make ARCH=arm menuconfig`

Noyau Linux – Compilation

- Pour compiler le noyau :
  - `make`
  - `make -j 4`
- Compilation croisée :
  - `make ARCH=arm CROSS-COMPILE=arm-linux-`

- Les binaires générés sont :
  - `vmlinux` : noyau non compressé
  - `arch/<arch>/boot/zImage` : noyau compressé (zlib)

- Installation (sur PC) :
  - `make install && make modules_install`

- Nettoyage :
  - `make clean` : force la recompilation des drivers
  - `make mrproper` : efface tous les fichiers générés (`.config` compris !)

Noyau Linux – Création d'un initramfs

- Sélectionner `CONFIG_INITRAMFS_SOURCE` dans la configuration

- Spécifier soit :
  - Une archive cpio existante
  - Un répertoire à archiver (contenant les répertoires, commandes et scripts nécessaire au démarrage)
  - Un fichier normal contenant une liste de commandes

```
dir /dev 755 0 0
nod /dev/console 644 0 0 c 5 1
dir /bin 755 1000 1000
slink /bin/sh busybox 777 0 0
file /bin/busybox /stuff/busybox 755 0 0
dir /proc 755 0 0
dir /sys 755 0 0
dir /mnt 755 0 0
file /init /stuff/init.sh 755 0 0
```

  - Liste des commandes dans `usr/gen_init_cpio.c`

Noyau Linux – Création d'un initramfs (suite)

- La dernière action d'un initramfs est le script `/init`

- Exemple :

```
#!/bin/sh
mkdir /proc /dev /root /var/log -p
ifconfig usb0 192.168.0.206 netmask 255.255.255.0
mknod /dev/mmcblk0p2 b 254 2
mkdir -p /mnt/realroot
mount /dev/mmcblk0p2  /mnt/realroot
exec switch_root /mnt/realroot /sbin/init 3
```

- Un initramfs n'est pas nécessaire si tous les drivers requis au démarrage et montage du rootfs sont compilés dans le noyau

Noyau Linux – Paramètres

- Le noyau peut recevoir des paramètres passés en arguments (comme un programme C avec argc/argv)
- Les paramètres sont à spécifier dans la configuration du boot loader
- Exemple :
  - `root=/dev/ram0 rw init=/linuxrc console=ttyS0,115200n8 console=tty`
  - `ramdisk_size=8192 cachepolicy=writethrough`
- Tout les paramètres sont détaillés dans `Documentation/kernel-parameters.txt`

Création d'un root FS

Root FS – Introduction

- Le système de fichiers racine, ou root FS, est le système de fichiers qui est monté à la fin de l'initialisation du noyau.
- Le contenu du root FS doit être créé sur une machine de développement (système host), puis archivé en un seul fichier qui sera copié dans le système cible.
- Les binaires contenus dans le root FS devront donc avoir été compilés pour l'architecture cible (donc compilation croisée)

Root FS – Contenu

- En général, le contenu d'un root FS pour un système embarqué est le plus minimaliste possible.
- `/dev` (si statique) ne contient que les entrées strictement nécessaires
- `/sbin`, `/bin`, `/usr/bin` ne contiendront que les commandes et programmes nécessaires
- `/lib` et `/usr/lib` ne contiendront que les librairies nécessaires
  - Utiliser `ldd` pour connaitre les librairies requise par un programme
- `/var` et `/tmp` peuvent éventuellement être des systèmes ramfs

Root FS – Choix

- Les contraintes du système cible imposent de faire des choix et/ou concessions :

- Librairie C :
  - Disponibilité pour l'architecture visée
  - Licence
  - Taille / options de configuration
  - Stabilité / maturité / sécurité / support
  - Support C++
  - Support threads NTPL ou LinuxThreads
  - Support de langages (i18n, l10n)

- Busybox ou commandes normales (binaires séparés) ?

Root FS – Choix (suite)

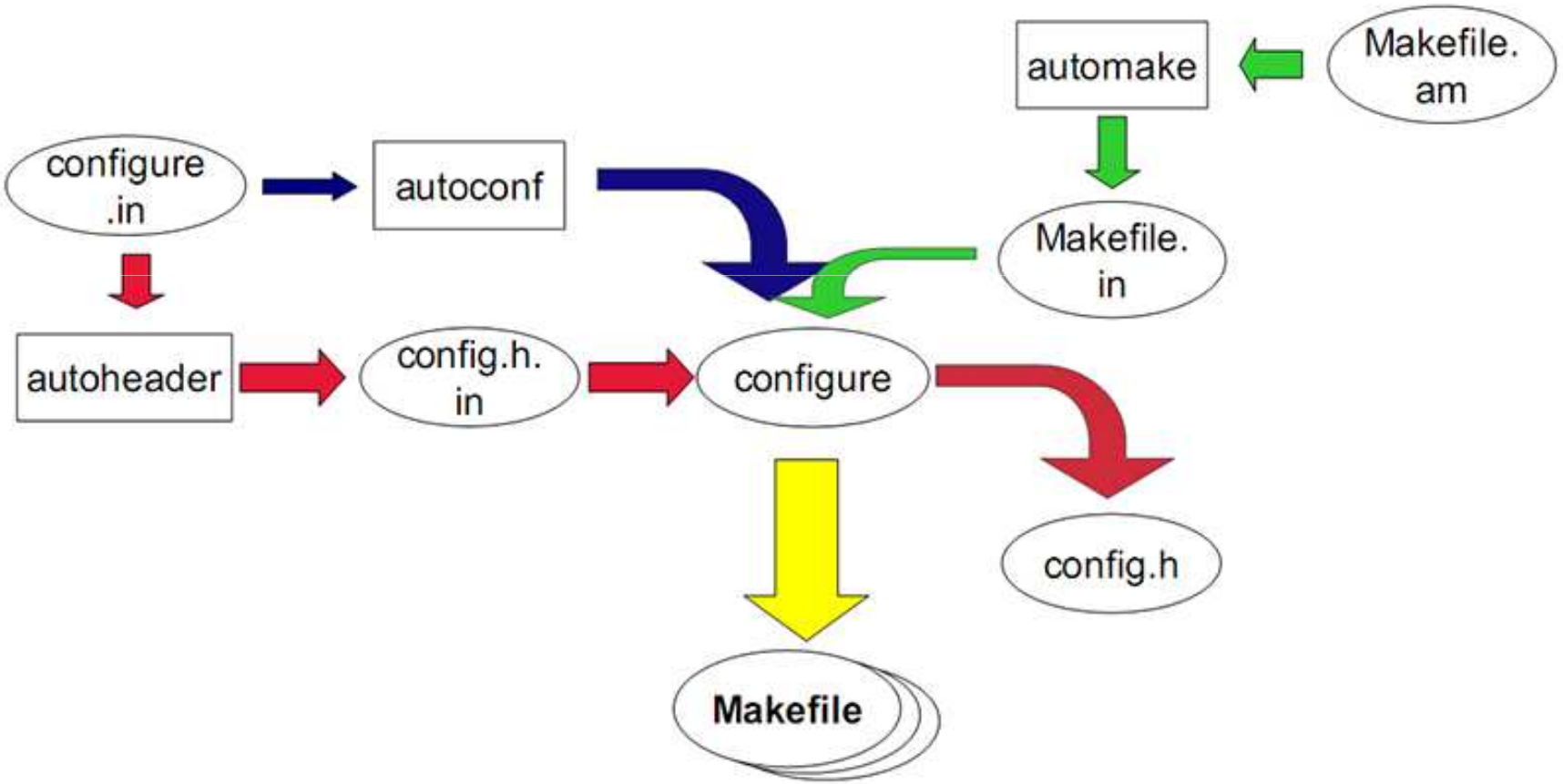
- Structure de l'initialisation du système :
  - Structure type System V
    - Beaucoup de scripts dans `/etc/init.d/rcX`
    - Les scripts requièrent souvent grep, sed, awk, etc.
    - Support des "runlevels"
    - Beaucoup d'option dans `/etc/inittab`
    - Beaucoup de fork et exec
  - Structure type Busybox
    - `/etc/inittab` beaucoup plus simple
    - Un seul `/etc/rcS` simple qui permet de lancer d'autre programmes
    - Pas de "runlevels"

Root FS – Dépendances

- Dépendances verticales :
  - Un paquet à besoin d'un autre pour fonctionner
    - Exemple : GTK-FB requiert DirectFB
- Dépendances diagonales :
  - Une spécificité du produit peut requérir d'autres paquets :
    - Exemple : Supporter LDAP implique sans doute d'avoir OpenLDAP inclus dans le système
- Dépendances de la construction :
  - Outils et librairies nécessaire à la compilation
- Dépendances d'exécution
  - Librairies, commandes, services

Root FS – Autotools

- Autotools est un ensemble d'outils de build pour assurer la portabilité d'un paquet



Root FS – Autotools (suite)

- Les scripts ./configure sont utilisés pour analyser l'environnement de développement courant :
  - Architecture : x86, PowerPC, etc
  - Librairies présentes : glib, libmysql, etc.
- Ces informations sont utilisées pour générer un Makefile
- Par défaut, ./configure réalise les tests en exécutant des petits programmes (donc incompatibles avec une compilation croisées)
- ./configure supporte la compilation croisée :

```
./configure --build=i686-pc-linux-gnu --host=m68k-coff
```

- Il faut que les scripts prévoient la compilation croisée
  - Il est parfois nécessaire de changer les scripts pour y ajouter ce support

Root FS – Construction

- La construction d'un root FS peut se faire :

- Manuellement
  - Pas reproductible
  - Risque d'erreurs
  - Gestion des dépendances difficiles

- Par un ensemble de scripts maison
  - Beaucoup de travail pour écrire et tester les scripts
  - Gestion des dépendances difficiles

- Avec un outil comme BuildRoot ou BitBake
  - Assure la reproductibilité
  - Gère les dépendances d'un nombre de paquets
    - Tout les paquets nécessaires ne sont pas forcément inclus

Root FS – Recettes

- Pour assurer la reproductibilité d'un processus de construction, on utilise une ou un ensemble de recettes

- Une recette contient les information pour :
  - Gérer les dépendances
  - Si le paquet n'est pas précompilé :
    - Eventuellement appliquer des patches
    - Procéder à la compilation
  - Procéder à l'installation du programme
- Exemples de recettes : RPM, Deb, ebuild

- Typiquement, un outil existe pour gérer un ensemble de recettes
  - Exemples : yum, dpgk, emerge, etc.
  - Pour l'embarqué : BuildRoot, Scratchbox, RPMbuild, BitBake

Busybox

Environnements – Busybox

- Busybox regroupe la plupart des utilitaires Unix en un seul exécutable

- Taille réduite :
  - Compilé statiquement avec uClibc : 500 KB
  - Compilé statiquement avec glibc : 1 MB

- Configuration des fonctionnalités souhaitées facile
- Gestion de paquet avec dpkg
- Implémentation légère de udev (mdev)
- Excellent pour des scripts complexes initramfs ou initrd

- Inclue un serveur DHCP et un serveur HTTP

- Licence GPL

- http://www.busybox.net

Busybox – Configuration

- Sources officielles de Busybox : busybox.net
- La configuration de Busybox est sauvée dans le fichier `.config` (situé à la racine des sources)
- L'outil de configuration est similaire à celui du noyau :
  - `make menuconfig`
  - `make xconfig`
  - `make gconfig`
- Configurations par défaut :
  - `make defconfig` : configuration avec les options les plus courantes
  - `make allnoconfig` : désélectionne toutes les options

Busybox – Compilation

- Sélectionner la chaine de compilation croisée
- Choisir le répertoire d'installation
- Sélectionner les utilitaires compilés nécessaires
- Ajouter le path de la chaine de compilation croisée au path courant du shell :
  - `export PATH=/usr/local/arm/3.3.2/bin:$PATH`
- Compiler Busybox :
  - `make`

Busybox – Compilation

Options	Name	N	M	Y	Value
Busybox Settings					
General Configuration					
Build Options					
Build BusyBox as a static binary (no shared libs)	STATIC	N		_	N
Force NOMMU build	NOMMU	N		_	N
Build shared libbusybox	BUILD_LIBBUSYBOX	N		_	N
Build with Large File Support (for accessing files > 2 GB)	LFS	N		_	N
Cross Compiler prefix	CROSS_COMPILER_PREFIX				
Additional CFLAGS	EXTRA_CFLAGS				
Debugging Options					
Installation Options					
Don't use /usr	INSTALL_NO_USR	N		_	N
Applets links					as soft
BusyBox installation prefix	PREFIX				./_inst
Busybox Library Tuning					
Applets					
Archival Utilities					

FORMATION

Busybox – Compilation

- Installer Busybox (créer les liens symboliques)
  - `make install`
  - Par défaut : `_install` dans le répertoire des source

```
-rwxr-xr-x    1 0        0          1065308 busybox
lrwxrwxrwx    1 0        0                7 init -> busybox
lrwxrwxrwx    1 0        0               12 ash -> /bin/busybox
lrwxrwxrwx    1 0        0               12 cat -> /bin/busybox
lrwxrwxrwx    1 0        0               12 chmod -> /bin/busybox
lrwxrwxrwx    1 0        0               12 cp -> /bin/busybox
lrwxrwxrwx    1 0        0               12 dd -> /bin/busybox
lrwxrwxrwx    1 0        0               12 echo -> /bin/busybox
```

Busybox – /sbin/init

- Busybox peut fournir `/sbin/init`
  - `/sbin/init` est un lien symbolique vers `/bin/busybox`

- Busybox utilise `/etc/inittab` :
  - Format classique : `id:runlevel:action:process`
  - `id` : spécifie le tty associé au programme, si non renseigné, Busybox utilise la console système
  - `runlevel` : ignoré, pas de runlevels
  - `action` : `sysinit`, `respawn`, `askfirst`, `wait`, `once`, `ctrlaltdel`, `shutdown`, `restart`
  - process : programme à exécuter

- Des scripts d'initialisation ou des applications spécifiques peuvent être ajouté au démarrage du système

Busybox – /etc/inittab

- Exemple de `/etc/inittab` :

```
::sysinit:/etc/init.d/rcS
::respawn:/sbin/getty 115200 ttyS0
::respawn:/usr/bin/my_app
::restart:/sbin/init
::shutdown:/bin/umount -a –r
```

- `/etc/init.d/rcS` : script d'initialisation du système
- `/sbin/getty` : lance une session de login sur un port série
- `/usr/bin/my_app` : application spécifique ajoutée
- `/sbin/init` : re-exécute init si le processus init redémarre
- `/bin/umount` : démonte les systèmes de fichiers lors du shutdown

Busybox – /etc/init.d/rcS

- Exemple de `/etc/init.d/rcS` :

```
#!/bin/sh

# Remount the root filesystem in read-write
# requires /etc/fstab
mount -n -o remount,rw /
# Mount /proc filesystem
mount /proc

# Start the network interface
/sbin/ifconfig eth0 192.168.172.10
```

- Note : aucuns des programmes ou scripts spécifiés dans `/etc/inittab` n'est lancé tant que `/etc/init.d/rcS` n'est terminé (action `sysinit`)

Buildroot

Buildroot – Configuration

- Le programme de configuration de Buildroot utilise la même interface que le noyau
  - `make menuconfig` ou `make xconfig`
- La configuration est enregistrée dans un fichier .config
- Le processus de configuration consiste à sélectionner :
  - L'architecture cible (ARM, MIPS, etc.)
  - Ses variantes (ARM7TDMI, Xscale, etc.)
  - L'ABI (ARM, EABI, OABI)
  - Les options de la cible
  - Les options de compilation (outils, options de gcc, debugging, répertoires, etc.)
  - Les paquets à inclure et leurs options

Buildroot – Configuration (suite)

- Buildroot peut générer une chaine de compilation croisée (si basée sur uClibc) ou en utiliser une déjà installée

- Si Builroot génère la chaine de compilation, il est possible de spécifier :
  - La version des headers du noyau
  - La version de uClibc et sa configuration
  - La version de gcc et ses options
  - La version de gdb et ses options
  - Les options générales de la chaine (IPv6, locale, etc.)

- Builroot peut compiler un noyau
  - La version, configuration, patches additionnels, format binaire (zImage, bzImage, uImage) peuvent être spécifiés

Buildroot – Paquets

- Buildroot permet d'inclure des centaines de paquets logiciels
  - Focus sur les logiciels pour l'embarqué
  - Gère les dépendances entre paquets

- Paquets principaux :
  - busybox, bash, bzip2, diffutils, flex, native toolchain, grep, bootutils, cups, at, beecrypt, dash, file, gamin, less, lsof, ltrace, memstat, module-init-tools , procps, psmisc, screen, strace, sudo, syslogd, klogd, util-linux, which, etc.

- Librairies principales :
  - libconfig, libconfuse, libdaemon, libelf, libevent, libgcrypt, libiconv, libidn, liblockfile, liboil, libsysfs, etc.

Buildroot – Paquets (suite)

- Base de données
- Editeur de texte
- Outils réseau
- Outils systèmes
- Programmes audio / vidéo
- Librairies graphiques
- Outils de compression / décompression
- Gestionnaires de paquets
- Langages interprétés
- Etc.

Buildroot – Répertoires

- Répertoire dans les sources de Buildroot :

`docs/`	Documentation
`package/`	Recettes pour compiler les paquets
`scripts/`	Utilitaires divers
`target/linux/`	Recettes pour compiler le noyau
`target/<fs>/`	Recettes pour générer les systèmes de fichiers
`target/device/`	Configurations de boards standards
`toolchain/`	Configurations pour compiler la chaine de compilation croisée
`dl/`	Sources téléchargées

Buildroot – Répertoires (suite)

- Répertoires générés dans `output/` :

`images/` Contient les images noyau, bootloader, root fs

`build/` Où tous les binaires (sauf la chaine de compilation) sont compilés. Contient un répertoire par paquet.

`staging/` Contient une hiérarchie similaire à un système racine, avec la chaine de compilation, les librairies

`target/` Contient presque le root FS pour la cible (sauf /dev)

`host/` Contient les outils compilés nécessaires à Buildroot

`toolchain/` Contient les composants de build de la chaine de compilation

Buildroot – Compilation

- Le plus simple :
  - `make`

- Des variables d'environnement peuvent être utilisées :
  - `HOSTCC` : le compilateur a utiliser
  - `LINUX26_CONFIG_FILE` : le .config du noyau a utiliser
  - `BUSYBOX_CONFIG_FILE` : le .config de BusyBox a utiliser
  - `UCLIBC_CONFIG_FILE` : le .config de uClibc a utiliser
  - `BUILDROOT_COPYTO` : répertoire où les images seront copiées

- Exemples :
  - `make UCLIBC_CONFIG_FILE=$HOME/uclibc.config`
  - `make HOSTCC=gcc-4.3-HEAD`
  - `make BUILDROOT_COPYTO=/tftpboot`

Buildroot – Ajouter un paquet

- Il est possible d'ajouter des paquets logiciels en plus de ceux gérés par défaut par Buildroot

- Créer un nouveau répertoire : `package/myapp/`

- Ajouter un fichier `Config.in` dans le répertoire, décrivant les options de configuration :

```
config BR2_PACKAGE_MYAPP
    bool "myapp"
    select BR2_PACKAGE_PKGCONFIG
    help
        This is a comment that explains what myapp is.
        http://www.myapp.org/
```

- Insérer ce nouveau `Config.in` dans le fichier `package/Config.in`

```
source "package/myapp/Config.in"
```

Buildroot – Ajouter un paquet (suite)

- Créer le fichier myapp.mk qui décrit les étapes de téléchargement, configuration, compilation et installation

- Exemple pour un paquet générique, non basé sur autotools :

```
MYAPP_VERSION = 1.1.0
MYAPP_SOURCE = myapp-$(MYAPP_VERSION).tar.gz
MYAPP_SITE = http://www.myapp.org/download
MYAPP_INSTALL_STAGING = NO
MYAPP_INSTALL_TARGET = YES
MYAPP_DEPENDENCIES = uclibc pkgconfig libgtk2
define MYAPP_BUILD_CMD
   $(MAKE) CC=$(TARGET_CC) LD=$(TARGET_LD) –C $(@D) all
endef
define MYAPP_INSTALL_TARGET_CMDS
   cp –dpf $(@D)/myapp $(TARGET_DIR)/usr/bin
endef
$(eval $(call GENTARGETS,package,myapp))
```

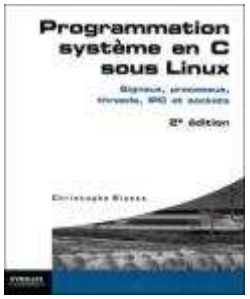
Buildroot – Ajouter un paquet (suite)

- Exemple pour un paquet basé sur autotools :

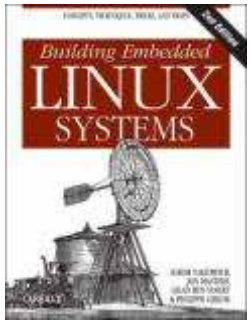
```
MYAPP_VERSION = 1.1.0
MYAPP_SOURCE = myapp-$(MYAPP_VERSION).tar.gz
MYAPP_SITE = http://www.myapp.org/download
MYAPP_INSTALL_STAGING = NO
MYAPP_INSTALL_TARGET = YES
MYAPP_DEPENDENCIES = uclibc pkgconfig libgtk2

$(eval $(call AUTOTARGETS,package,myapp))
```

Bibliographie

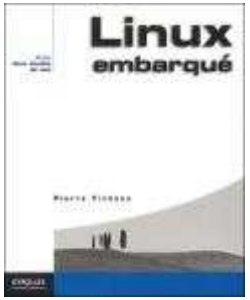
Bibliographie



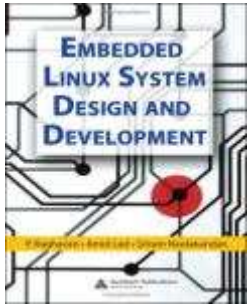
Editeur : Eyrolles
Auteur : C. Blaess
Edition : 2e édition
Nb de pages : 964 pages
Langue: Français
Notation: ★ ★ ★ ★ ★



Editeur : O'Reilly
Auteur : Karime Yaghmour
Parution : 02/09/2005
Edition : 2e édition
Nb de pages : 462 pages
Langues: Anglais
Notation: ★ ★ ★ ★ ★

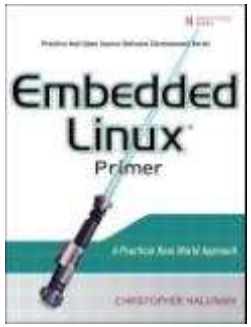


Editeur : Eyrolles
Auteur : P. Ficheux
Parution : 29/09/2005
Edition : 2e édition
Nb de pages : 330 pages
Langues: Anglais
Notation: ★ ★ ★ ★ ★



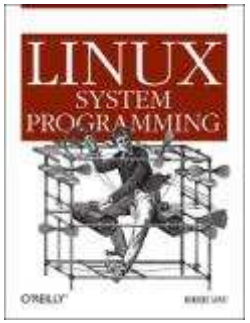
Editeur : Auerbach Publishers
Auteur : Amold Lad
Parution : 21/12/2005
Edition : 2e édition
Nb de pages : 432 pages
Langues: Anglais
Notation: ★ ★ ★ ★ ★

Editeur : Dunod
Auteur : J. Delacroix
Parution : 11/02/2009
Edition : 3e édition
Nb de pages : 348 pages
Langues: Anglais
Notation: ★ ★ ★ ★

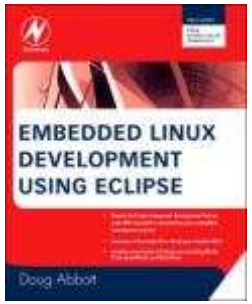


Editeur : Prentice Hall
Auteur : Christophe Alinan
Parution : 28/02/2006
Edition : 2e édition
Nb de pages : 432 pages
Langues: Anglais
Notation: ★ ★ ★ ★ ★

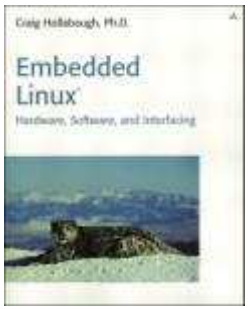
Bibliographie (suite)



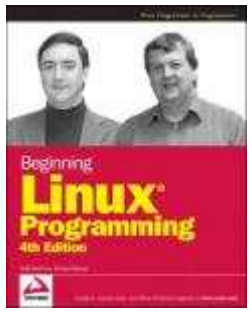
Editeur : O'Reilly
Auteur : Robert Love
Parution : 02/10/2007
Nb de pages : 368 pages
Langue: Anglais
Notation: ★ ★ ★ ★



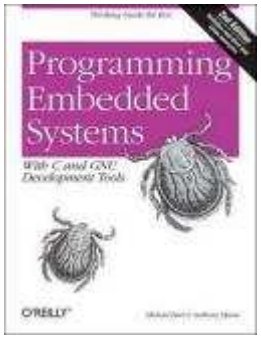
Editeur : Newnes
Auteur : Doug Abbott
Parution : 26/12/2008
Nb de pages : 264 pages
Langue: Anglais
Notation: ★ ★ ★ ★



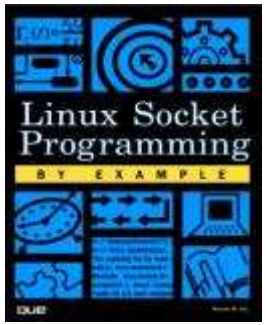
Editeur: Addison-Wesley
Auteur : Craig Hollabaugh
Parution : 06/10/2002
Nb de pages : 432 pages
Langue: Anglais
Notation: ★ ★ ★ ★



Editeur : Wrox
Auteurs : Neil Matthew, Richard Stones
Parution : 10/2007
Nb de pages : 816 pages
Langue: Anglais
Notation: ★ ★ ★ ★



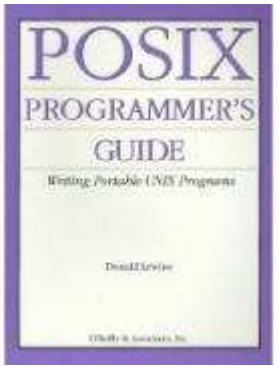
Editeur: O'Reilly
Auteur : M. Bar & A. Massa
Parution : 20/10/2006
Nb de pages : 301 pages
Langue: Anglais
Notation: ★ ★ ★ ★



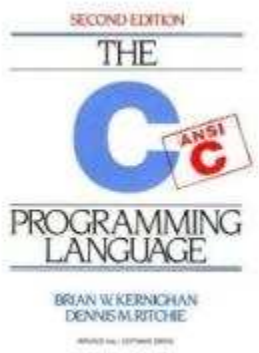
Editeur : Wrox
Auteurs : Warren Gay
Parution : 2 Mai 2000
Nb de pages : 576 pages
Langue: Anglais
Notation: ★ ★ ★ ★

Bibliographie (suite)

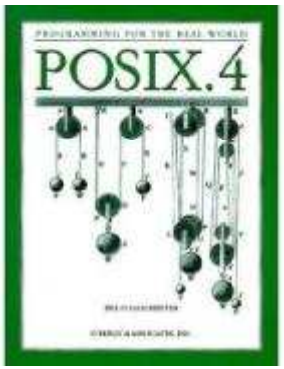
Editeur : Dunod
Auteur: A. Braquelaire
Collection : Sciences sup
Parution : 13/05/2005
Edition : 4e édition
Nb de pages : 652 pages
Notation: ★ ★ ★ ★



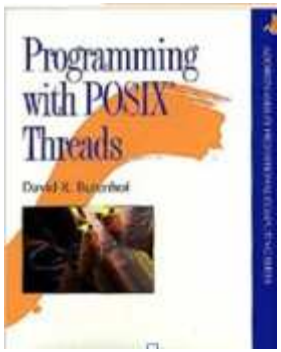
Auteur: Lewine
Editeur : O'Reilly
Parution: 28 mai 1991
Langue : Anglais
Nb de pages : 640 pages
Notation**:** ★ ★ ★ ★



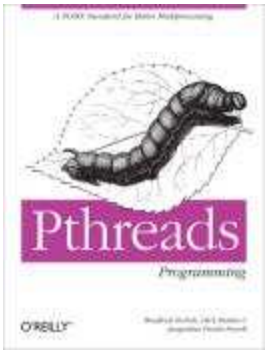
Editeur : Prentice Hall;
Auteurs:. Kernighan&Ritchie
Édition : 2 d edition
Parution: 1 avril 1988
Broché: 274 pages
Langue : Anglais
Notation**:** ★ ★ ★ ★ ★



Auteur: Gallmeister
Editeur : O'Reilly
Parution: 1 octobre 1994
Langue : Anglais
Nb de pages : 570 pages
Notation**:** ★ ★ ★ ★

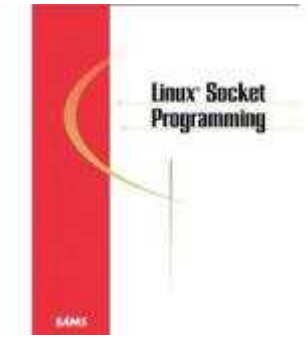


Editeur : Addison-Wesley
Auteurs: David Butenhoff,
Parution: 8 Aout 1997
Nb de pages : 400 pages
Langue : Anglais
Notation**:** ★ ★ ★ ★

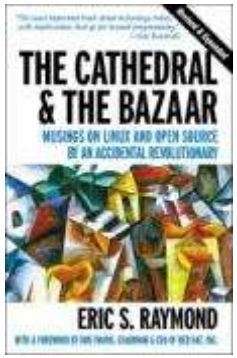


Auteur: Nichols
Editeur : O'Reilly
Parution: 1 octobre 1996
Langue : Anglais
Nb de pages : 288 pages
Notation**:** ★ ★ ★ ★

Bibliographie (suite)



Editeur : **Sams Publishing**
Auteurs:. Sean Walton
Parution: **8 février 2001**
Broché: 400 pages
Langue : Anglais
Notation**:** ★ ★ ★ ★



Un ouvrage pour méditer…
Editeur : **Sams Publishing**
Auteurs:. Sean Walton
Parution: **8 février 2001**
Broché: 400 pages
Langue : Anglais
Notation**:** ★ ★ ★
>lien gratuit en français<

Bibliographie intéressante sur: http://uuu.enseirb.fr/~kadionik/

Ouvrages gratuits sur le Net

- Programmation Avancée sous Linux (301 pages)
  Auteurs: Mark Mitchell Jeffrey Oldham Alex Samuel
  http://www.advancedlinuxprogramming-fr.org/lib/exe/fetch.php?id=Accueil&cache=cache&media=book-screen.pdf
  (version éditable)
  http://www.advancedlinuxprogramming-fr.org/lib/exe/fetch.php?id=Accueil&cache=cache&media=book-print.pdf
  (version imprimable)

- Cours Système (148 pages)
  Auteur: D.Revuz 17 février 2005
  http://www-igm.univ-mlv.fr/%7Edr/CS/