Matériel de réseau

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Chapitre 3

Linux embarqué

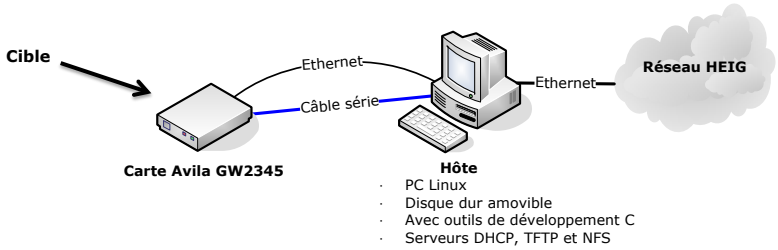
Contenu

Création d'un système Linux complet

1. Créer l'environnement de développement
2. Créer la 'toolchain' (compilateur, ...)
3. Construire le noyau Linux
4. Construire le système de fichiers racine
   - Utilitaires Linux et librairies
   - Fichiers de configuration (/etc)
   - Fichiers spéciaux (/dev)
5. Configurer le chargeur d'amorçage
6. Installation et test

Environnement de développement

- Hôte avec 1 interface série et 2 interfaces Ethernet
- Un disque dur amovible par groupe
  - Linux
  - Toolchain GNU C
  - Serveurs DHCP et TFTP



Structure des répertoires

Répertoire	Contenu
$PRJROOT	Racine de l'arborescence du projet, p.ex. */home/labo/MAR2012/*
├ *bootldr*	Chargeur d'amorçage (*boot loader*)
├ *images*	**Images binaires du noyau et du système de fichiers racine**
├ *kernel*	**Le ou les sources des différents noyaux Linux**
├ *rootfs*	**Le système de fichiers racine complet**
├ *sysapps*	**Composants du système d'exploitation qui doivent être compilées**
├ *tools*	**Les outils de développement C de GNU (GCC, binutils, …)**
├ *tmp*	Répertoire pour des fichiers temporaires
├ *doc*	**Documentation du projet**
├ *devel*	Le code source des logiciels développés par nous-mêmes

Documentation du projet

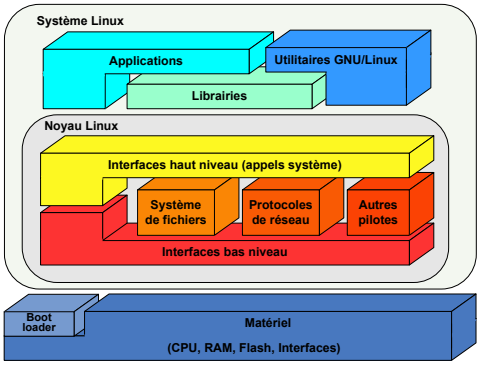
- La documentation est très importante dans un projet Linux embarqué
  - Configurations complexes
  - Interdépendance des composants (versions, configuration)
  - Souvent une approche par tâtonnement
- A l'aide des formulaires de l'annexe du chapitre 2
  - Source: http://www.embeddedtux.org/worksheet.html

Définition du projet

Project identification	
Name: Abraham Rubinstein **Internal ID:** MAR2012 **Project leader:** Abraham Rubinstein	
Start date: 12 janvier 2013 **Expected completion date:** février 2013	
Project description: Développement d'un système Linux embarqué pour les cartes de développement Gateworks Avila GW2345. Le système embarqué comprendra : • Un noyau Linux • Un système de fichiers racine complet • Des scripts d'initialisation • Un chargeur d'amorçage configuré	
Type of system: Gateworks Avila GW2345	**Size:** Moyenne : 4 MB flash, 32 MB RAM
Time constraints: sans contraintes temps-réel	**Degree of user interaction:** Faible, à travers la liaison série
Are networking services offered or used? Non, les interfaces réseau ne seront pas encore fonctionnelles.	

Architecture d'un système Linux

- Chargeur d'amorçage (*boot loader*)
  - Démarre le noyau
- Noyau
  - Logiciel principal du système qui contrôle l'accès aux périphériques et qui contrôle les autres processus
- Fichiers spéciaux
  - Représentent les périphériques du système
  - P.ex. /dev/hda pour le disque dur
- Fichiers de configuration
  - Configuration du système, p.ex. mots de passe
- Applications, utilitaires, librairies



La 'toolchain' GNU

Toolchain

Outils de développement C/C++

- Compilateur C/C++ : GNU gcc
- Librairie standard C : glibc
  - Définit les « appels système » et autres fonctionnalités de base
- Utilitaires binaires : binutils
  - Éditeur de liens ld, assembleur as, archivage ar, ...

- La toolchain doit générer du code exécutable pour la cible
  - Compilateur croisé pour l'architecture de la cible
  - Librairies pour l'architecture de la cible

Compilation croisée

- Le compilateur croisé doit s'exécuter sur l'hôte mais générer du code pour la cible
  - Dans notre projet:
    - Architecture de l'hôte: Intel Pentium x86, Little Endian
    - Architecture de la cible: XScale/ARM, Big Endian
- Avantage:
  - Le développement peut se faire sur l'hôte
- Inconvénients:
  - Construction difficile de la toolchain croisée
  - Les exécutables ne peuvent pas être testés sur l'hôte

Quelle toolchain ?

- Toolchain commerciale (p.ex. MontaVista Linux)
  - Fournie par le distributeur commercial, ensemble avec la distribution Linux commerciale
  - Bonne qualité, testée, adaptée à la distribution
- Toolchain libre, pré-compilée (p.ex. Snapgear)
  - Choisir la bonne toolchain (architecture, Little/Big Endian)
  - Souvent mal documentée (options de compilation?)
- Construction manuelle de la toolchain
  - Maîtrise des options de configuration
  - Procédure difficile et sujette à l'erreur

Construction de la toolchain

- Relativement difficile et sujette à l'erreur
  - Versions incompatibles du compilateur, de la librairie standard C et du noyau Linux

Versions recommandés (cf. Dan Kegel, Yaghmour)

Outil	Dernière version (oct. 2005)	Version recommandée				
		x86	ARM, PowerPC	MIPS	MIPS-el	SuperH
gcc	3.4.4	3.4.4 ou 4.0.1	3.4.4	egcs-1.1.2	3.2.1	3.0.1
glibc	2.3.5	2.3.5	2.3.5	2.0.6	2.3.1	2.2.4
binutils	2.16.1	2.15	2.15	2.8.1	2.14	2.14
Noyau Linux	2.6.13.2	2.6.11.3	2.6.8	2.4.x	2.4.x	2.4.x

Procédure de la construction

1. Installation des fichiers header du noyau
   - Pour les appels systèmes depuis la librairie C
2. Compilation des utilitaires binaires
   - Doivent pouvoir manipuler du code objet / exécutable de la cible
3. Première compilation de gcc
   - Sera capable de générer du code pour la cible
4. Compilation de la glibc
   - Librairies résultant seront prêtes pour cible
5. Deuxième compilation de gcc
   - Support pour d'autres langages que C, nécessite glibc

Outil Crosstool

- Permet d'automatiser la construction de la toolchain
  - Gestion des versions compatibles de gcc, glibc, Linux
  - Télécharge les sources
  - Applique des correctifs
  - Compile, teste et installe la toolchain

  - Configuration simple
  - Toolchain de bonne qualité, avec les correctifs connus

- Source : http://kegel.com/crosstool

Installation de la toolchain

Répertoire	Contenu
$PROJROOT/*tools/gcc-3.4.4-glibc-2.3.5/armeb-xscale-linux-gnu*	Répertoire racine de la toolchain
├ ***bin***	**Les exécutables du compilateur croisé et des utilitaires binaires.**
├ *include*	Les fichiers header pour le compilateur croisé
├ *lib*	Les librairies pour la compilation croisée
├ ***armeb-xscale-linux-gnu***	**Ce répertoire contient des fichiers qui sont nécessaires sur la cible**
├ *bin*	Exécutables et scripts pour la cible
├ *etc*	Fichiers de configuration à copier dans le répertoire */etc* de la cible
├ *include*	Fichiers header pour la cible
├ ***lib***	**Les librairies de la GLIBC, à copier dans le répertoire */lib* de la cible**
├ *libexec*	Utilitaires annexes
├ *sys-include*	Mêmes fichiers header comme dans include, non nécessaire pour la cible

Vérification de la toolchain

- Version du compilateur

```
> armeb-xscale-linux-gnu-gcc -v
...
Configured with: $PROJROOT/tmp/crosstool-0.38/build/armeb-xscale-linux-gnu/gcc-3.4.4-glibc-2.3.5/
gcc-3.4.4/configure
--target=armeb-xscale-linux-gnu --host=i686-host_pc-linux-gnu
--prefix=$PROJROOT/tools/gcc-3.4.4-glibc-2.3.5/armeb-xscale-linux-gnu
--with-cpu=xscale --enable-cxx-flags=-mcpu=xscale
...
--enable-languages=c,c++ --enable-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.4.4
```

- Type de la librairie C
  - Avec la commande `file'

```
> file libc-2.3.5.so
libc-2.3.5.so: ELF 32-bit MSB shared object, ARM, version 1 (ARM), not stripped
```


Utilisation de la toolchain

- Exécutable du compilateur:
  armeb-xscale-linux-gnu-gcc
  - A ajouter aux chemins des exécutables
    export PATH=$PATH:$PROJROOT/tools/gcc-3.4.4-glibc-2.3.5/armeb-xscale-linux-gnu/bin

- Compilation :

```
> armeb-xscale-linux-gnu-gcc –o hello hello.c

> file hello
  hello: ELF 32-bit MSB executable, ARM, version 1 (ARM), for
  GNU/Linux 2.4.3, dynamically linked (uses shared libs)
```

Construction du noyau Linux

Choisir, configurer, compiler et installer un noyau Linux

Procédure de construction du noyau

1. Choix de la version du noyau
2. Téléchargement des sources
3. Application de correctifs
4. Configuration des options du noyau
   - Architecture de la cible
   - Fonctionnalités du système
   - Pilotes
5. Compilation
6. Sauvegarde du noyau et de la configuration

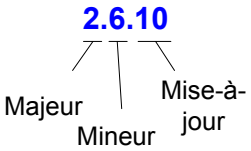
Le noyau Linux

- Le noyau officiel est géré par Linus Torvalds
  - Disponible sur www.kernel.org
- D'autres groupes de développeurs sont spécialisés à des architectures non x86
  - Ils testent le noyau pour leurs architectures et corrigent les bogues
  - Ils mettent à disposition des noyaux modifiés
  - Les modifications seront normalement intégrées plus tard dans le noyau officiel

Processeur	Source principale	Alternatif
x86	http://www.kernel.org (noyau officiel)	
ARM	http://www.kernel.org	http://www.arm.linux.org.uk/developer
PowerPC	http://www.kernel.org	http://penguinppc.org
MIPS	http://www.linux-mips.org	
SuperH	http://linuxsh.sourceforge.net	

Versions du noyau

- Numéros de version du noyau
  - Noyaux stables
    - Numéro mineur pair : 0, 2, 4, 6
  - Noyaux de développement
    - Numéro mineur impair: 1, 3, 5



- Depuis le noyau 2.6
  - Pas de noyau de développement 2.7
  - Quatre numéros, le dernier pour les corrections de bogues : 2.6.31.1

Quel est le 'bon' noyau pour mon projet ?

- Contrairement aux autres composants du système, le dernier noyau stable est souvent le meilleur choix
  - Le noyau est soigneusement testé et n'a que peu de bogues
  - Peu de dépendances entre la version du noyau et celles des autres composants
  - Bon support des périphériques
- La mise à jour du noyau pendant le projet est normalement possible sans introduire des incompatibilités

Préparation du noyau

- Téléchargement de la dernière version et installation des sources dans le répertoire $PROJROOT/kernel

```
> cd $PROJROOT/kernel
> wget http://mirror.switch.ch/ftp/mirror/kernel/
    linux/kernel/v2.6/linux-2.6.13.3.tar.bz2
--10:54:21--  http://mirror.switch.ch/ftp/mirror/kernel/linux/
              kernel/v2.6/linux-2.6.13.tar.bz2
       => `linux-2.6.13.tar.bz2'
Length: 38,372,729 [application/x-tar]
100%[====================================>] 38,372,729 1.62M/s ETA 00:00
> tar –xjvf linux-2.6.13.3.tar.bz
linux-2.6.13/
linux-2.6.13/COPYING
linux-2.6.13/CREDITS
linux-2.6.13/Documentation/
linux-2.6.13/Documentation/00-INDEX
...
```

Correctifs (*patches*)

- Un correctif modifie les fichiers de code source ou ajoute des fichiers
  - Corrections de bogues
  - Ajout de nouvelles fonctionnalités
- Un fichier 'patch' contient la différence entre le fichier original et le fichier modifié

Exemple

linux-orig/hello.c :

```
#include <stdio.h>
int main(void)
{
   printf("Hello!\n");
}
```

linux-modif/hello.c :

```
#include <stdio.h>
int main(void)
{
   printf("Bonjour!\n");
}
```

my.patch :

```
--- linux-orig/hello.c
+++ linux-modified/hello.c
@@ -1,5 +1,5 @@
 #include <stdio.h>
 int main(void)
 {
-   printf("Hello!\n");
+   printf("Bonjour!\n");
 }
```


Application de correctifs

- Exemple

```
> cd $PROJROOT/kernel
> ls
linux-2.6.13.3
patch-2.6.13-rc7-ds1
> cd linux-2.6.13.3
> patch -p1 < ../patch-2.6.13-rc7-ds1
patching file arch/arm/Kconfig
patching file arch/arm/Makefile
patching file arch/arm/boot/compressed/Makefile
patching file arch/arm/boot/compressed/head.S
patching file arch/arm/configs/ixdp2351_defconfig
...
```

Configuration du noyau

Adaptation du noyau aux besoins spécifiques
- Important pour réduire la taille du noyau

- Choix des fonctionnalités du noyau
  - P.ex. pilotes des cartes réseau
  - Fonctionnalités comme IPSec, lecteur réseau
- Configuration de base
  - Architecture, plateforme, Little/Big Endian

- Crée un fichier de configuration « .config » avec toutes les options de configuration

Méthodes de configuration

- make xconfig ou make gconfig
  - Configuration à l'aide d'une interface graphique
- make menuconfig
  - Configuration à l'aide d'un menu hiérarchique
- make config
  - Sans possibilité de navigation, pose une question pour chaque option de configuration

- make oldconfig
  - Utilise un fichier .config existant et ne pose de questions que pour les options qui manquent

Compilation pour l'architecture de la cible

- Utilisation du compilateur croisé
  - Paramètre CROSS_COMPILE de la commande make
  - Indique le préfixe à ajouter devant 'gcc'
- Choix de l'architecture du noyau
  - Paramètre ARCH de la commande make

Exemple

Architecture	Description
alpha	Processeurs DEC Alpha
arm	Pour processeurs modernes ARM et XScale
i386	Processeurs x86
ia64	Intel Itanium, à 64 bit
mips	MIPS
ppc	PowerPC
ppc64	PowerPC à 64 bit
sh	Motorola SuperH
sparc	SUN SPARC
sparc64	SUN SPARC à 64 bit
x86_64	AMD Athlon-64 et Opteron

```
> make ARCH=arm CROSS_COMPILE=armeb-xscale-linux-gnu- menuconfig
```

- Utilise le compilateur *armeb-xscale-linux-gnu-gcc* et l'implémentation dans le répertoire linux-2.6.13.3/arch/arm

Options de configuration

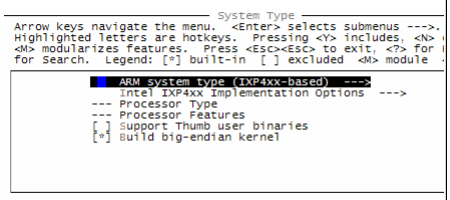
Menu	Explications
Code maturity level options	Choix de composantes expérimentales. Non nécessaire.
General setup	Options générales. Utiliser les valeurs par défaut.
Loadable module support	Important. Configuration du support pour des modules dynamiques.
System Type	**Très important.** Choix de l'architecture.
Bus support	Le support pour le bus PCI doit être activée.
Kernel Features	Fonctionnalités supplémentaires du noyau. Utiliser les valeurs par défaut.
Boot options	**Très important.** Il faut configurer la ligne de commande du noyau.
Floating point emulation	Choisir une des émulations. Utiliser les valeurs par défaut.
Userspace binary formats	Le support pour les exécutables en format ELF est obligatoire.
Power management options	Options de la mise en veille de l'ordinateur. Non nécessaire sur un dispositif de réseau.
Networking	**Important.** Dans ce menu et les sous-menus il est possible de configurer les fonctionnalités de réseau, par exemple si IPv6 doit être inclus ou des fonctionnalités de filtrage (firewall).
Device Drivers	**Important**. Configuration des pilotes à inclure.
File systems	**Très important.** Support pour différents systèmes de fichiers.
Profiling support	Fonctions spécialisées pour la mesure des performances du système. Non nécessaires.
Kernel hacking	Fonctions de débogage du noyau. Sélectionner si souhaité pendant le test du système.
Security options	Fonctions avancées de sécurité. Non nécessaires.
Cryptographic options	Fonctions cryptographiques à utiliser par le noyau.
Library routines	Des fonctions CRC (codes cycliques) supplémentaires. Normalement pas nécessaires.

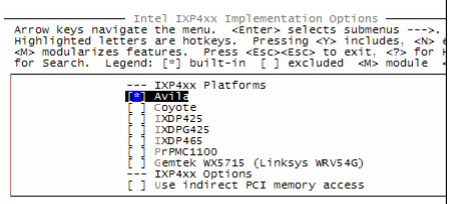
Menu des pilotes

Pilotes	Explication
Memory Technology Devices (MTD)	**Important**. Support pour les dispositifs de stockage, notamment les mémoires flash.
Plug and Play support	Pilotes pour des périphériques Plug-and-Play. Normalement pas nécessaire sur des systèmes embarqués.
Block devices	**Important**. Il faut notamment configurer un disque virtuel en RAM.
ATA/ATAPI/MFM/RLL support	Support pour des disques durs. Normalement pas nécessaire sur les systèmes embarqués.
SCSI device support	Support pour des interfaces SCSI, utilisées notamment par des disques durs SCSI, mais utilisées aussi pour les clés mémoire USB. Normalement pas nécessaire sur un système embarqué.
Network device support	**Important.** Pilotes des cartes de réseau. Il faut choisir notamment les cartes Ethernet et Wireless LAN.
Input device support	Support pour la souris, le clavier, etc. Normalement pas nécessaire sur un système embarqué.
Character devices	Support pour les consoles de commande virtuelles. Nécessaire pour un système embarqué. Utiliser les options par défaut.
I2C support	I²C est un bus série interne à bas débit. Il est souvent utilisé sur des plateformes embarqués. Inclure si nécessaire.
Multimedia devices	Pilotes pour des périphériques vidéo (non moniteur). Non nécessaire.
USB support	Support pour les interfaces USB. Sera discuté dans la partie II du cours.
MMC/SD Card support	Support pour des cartes de mémoire MMC ou SD. Normalement pas nécessaire.

Configuration pour la plateforme cible

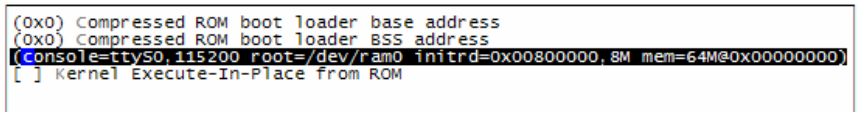
- Le noyau Linux offre du support pour beaucoup de plateformes spécifiques
  - Exemple PC normal :
    - Plateforme « PC-compatible »
    - Processeur Pentium, Athlon, ...
- Notre plateforme
  - System Type:
    - ARM → IXP4xx-based
    - Big Endian
  - Implementation Option:
    - Avila





Ligne de commande du noyau

- Permet de spécifier des options supplémentaires lors du démarrage du noyau



Options typiques

- Redirection de la console sur le port série
  - console=ttyS0,115200 ('S' majuscule !)
- Quantité de mémoire disponible
  - mem=64M@0x00000000
- Options du ramdisk
  - root=/dev/ram0 initrd=0x00800000,8M

Disques mémoire (RAMDISK)

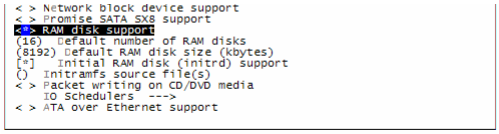
- Dans un PC normal, le OS se trouve sur le disque dur
  - Lecture et modification des fichiers du système
- Les systèmes embarqués ne possèdent souvent pas de disque dur
- La mémoire Flash est souvent utilisée en lecture seule

Utilisation d'un ramdisk

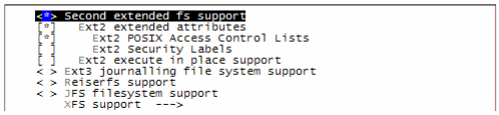
- Le chargeur d'amorçage copie l'arborescence de l'OS de la mémoire Flash et en mémoire RAM, dans un ramdisk
- Le noyau est configuré pour trouver et utiliser le ramdisk
- Le ramdisk est utilisé comme un disque dur normal pendant l'opération du système
- Les modifications des fichiers sont perdues lors du redémarrage du système

Configuration d'un ramdisk

- Activation d'un ramdisk initial
  - Menu « Block devices »



- Pilote du système de fichiers
  - Menu « File systems »
  - P.ex. ext2, comme pour une partition Linux normale



Compilation du noyau

- Pour le noyau 2.4 (inutile pour 2.6)

```
make ARCH=arm CROSS_COMPILE=armeb-xscale-linux-gnu- dep
```

- Pour compiler le noyau et les modules

```
make ARCH=arm CROSS_COMPILE=armeb-xscale-linux-gnu- all
```

  - Génère le noyau *arch/arm/boot/zImage*
    - Sur les PC, on utilise un autre format du noyau: bzImage
  - Génère les modules

Erreurs de compilation du noyau

- Causes possibles
  - Mauvaise toolchain avec une version incompatible du compilateur
    - Vérifier versions et options de la toolchain
  - Omission d'une fonction ou d'un pilote ou choix d'une option qui est incompatible avec l'architecture
    - Regarder le message d'erreur
    - Corriger la configuration
    - Parfois il est nécessaire d'utiliser make clean pour effacer les résultats de la compilation précédente
  - Bogue du noyau
    - Chercher dans les forums et newsgroups du kernel

Sauvegarde du noyau

- Si la compilation aboutit, le noyau compilé se trouve dans *arch/arm/boot/zImage*
- Sauvegarde:
  - Noyau zImage
  - Fichier de configuration .config
  - Fichier System.map (symboles du noyau)
- Utiliser des noms parlants

```
> cp -i arch/arm/boot/zImage   $PROJROOT/images/2.6.13.3-chapitre3.zImage
> cp -i .config                $PROJROOT/images/2.6.13.3-chapitre3.config
> cp -i System.map             $PROJROOT/images/2.6.13.3-chapitre3.System.map
```

Sauvegarde des modules

- Crée toute une arborescence de répertoires avec les pilotes du noyau
- Cette arborescence sera copiée dans le répertoire /lib sur la cible

Sauvegarde

```
> make ARCH=arm CROSS_COMPILE=armeb-xscale-linux-gnu-
INSTALL_MOD_PATH=$PROJROOT/images/2.6.13.3-chapitre3.modules/
modules_install
```

- Affiche beaucoup de message d'erreurs
- Devrait créer le fichier modules.dep qui contient les dépendances entre les modules
- Comme les modules sont en format pour l'architecture ARM, ils ne peuvent pas être traités correctement
- Le fichier modules.dep sera créé plus tard

Système de fichiers racine

- Librairies
- Fichiers spéciaux
- Utilitaires Linux
- Fichiers de configuration

Système de fichiers racine

- Arborescence des répertoires d'un système Linux
- Racine : répertoire '/'

Répertoire	Contenu
/bin	**Commandes utilisateur essentielles.**
/boot	Fichiers utilisés par le chargeur d'amorçage.
/dev	**Fichiers spéciaux qui représentent les périphériques du système, p.ex. /dev/usb.**
/etc	**Fichiers de configuration.**
/home	Répertoires des personnels des utilisateurs. Normalement pas présent sur un système embarqué.
/lib	**Librairies essentielles, comme la libraire C, et les modules du noyau.**
/mnt	Répertoire sous lequel on peut 'monter' d'autres systèmes de fichiers, comme par exemple le lecteur CDROM ou un lecteur disquette.
/proc	Système de fichiers spéciaux qui représentent des paramètres du noyau.
/root	Répertoire personnel de l'administrateur du système.
/sbin	**Commandes essentielles de l'administrateur.**
/tmp	Fichiers temporaires. (Avec les droits de lecture et d'écriture pour tout le monde).
/usr	**Arborescence qui contient les applications (*/usr/bin*) et leurs libraires (*/usr/lib*), ainsi que des commandes administrateur supplémentaires (*/usr/sbin*).**
/var	Données dynamiques qui sont écrites et lues pendant l'opération du système. Avec quelques sous-répertoires.

Librairies

- Nécessaires pour l'exécution des programmes sur la cible
  - Contiennent les fonctions que les applications peuvent appeler
- Seront copiées dans le répertoire /lib sur la cible
  - Doivent être compilées pour la cible
  - Se trouvent dans le répertoire
    armeb-xscale-linux-gnu/lib
    du compilateur croisé
  - 96 fichiers qui occupent 24 MB !

Types de librairies

```
> ls –l libm*
540K -rwxr-xr-x  libm-2.3.5.so
764K -rw-r--r--  libm.a
  0 lrwxrwxrwx  libm.so -> libm.so.6
  0 lrwxrwxrwx  libm.so.6 -> libm-2.3.5.so
```

Librairie dynamique (libm-2.3.5.so)
- Suffixe so: 'Shared object': librairie dynamique
- Doit être présente lors de l'exécution
- Nom contient le numéro de version complet

Lien avec la version majeure (libm.so.6)
- Souvent un logiciel ne s'intéresse qu'à la version majeure d'une libraire
- Les versions de la même version majeure sont compatibles entre elles
- Souvent aucune relation entre le numéro de version exacte (2.3.5) et la version majeure (6)
- Ce lien permet de trouver la librairie, connaissant la version majeure

Lien sans version (libm.so)
- Utilisée par des programmes qui n'ont pas besoin d'une version spécifique

Librairie statique (libm.a)
- Utilisée pour le linkage statique, lors de la compilation

Quelles librairies sont nécessaires ?

Uniquement les librairies dynamiques (librairie réelle et les liens)

Librairie/ Fichier	Contenu	Recommandation
ld	Éditeur de liens (*linker*).	Indispensable
libc	Librairie standard C	Indispensable
libcrypt	Fonctions cryptographiques.	Nécessaire p.ex. pour les protocoles d'authentification
libdl	Fonctions pour accéder à une librairie dyn.	Souvent nécessaire
libgcc_s	Fonctions bas niveau de GCC.	Indispensable
libm	Fonctions mathématiques.	Souvent nécessaire
libnss_compat	Fonctions NSS.	Utilisée par exemple des serveurs Web ou des services de nommage.
libnss_dns	Fonctions NSS.	Voir libnss_compat
libnss_files	Fonctions NSS.	Voir libnss_compat
libpthread	Threads POSIX.	Nécessaire pour les threads
libresolv	Fonctions DNS.	Utilisée pour la résolutions de noms
libutil	Fonctions de login.	Nécessaire
libstdc++	Librairies C++	Nécessaire pour C++, 3.8 MB

Installation des libraires

- Copier les librairies nécessaires et les liens symboliques
  - ...*armeb-xscale-linux-gnu/lib/* → *$PROJROOT/rootfs/lib*
- Résultat
  - 24 fichiers occupant 2.5 MB

Librairie uClibc

- Librairie standard C réduite pour les systèmes embarqués
- Même noms de librairies, même procédure d'installation
- Occupe 300 KB

Système de fichiers racine

- Arborescence des répertoires d'un système Linux
- Racine : répertoire '/'

Répertoire	Contenu
/bin	**Commandes utilisateur essentielles.**
/boot	Fichiers utilisés par le chargeur d'amorçage.
/dev	**Fichiers spéciaux qui représentent les périphériques du système, p.ex. /dev/usb.**
/etc	**Fichiers de configuration.**
/home	Répertoires des personnels des utilisateurs. Normalement pas présent sur un système embarqué.
/lib	**Librairies essentielles, comme la libraire C, et les modules du noyau.**
/mnt	Répertoire sous lequel on peut 'monter' d'autres systèmes de fichiers, comme par exemple le lecteur CDROM ou un lecteur disquette.
/proc	Système de fichiers spéciaux qui représentent des paramètres du noyau.
/root	Répertoire personnel de l'administrateur du système.
/sbin	**Commandes essentielles de l'administrateur.**
/tmp	Fichiers temporaires. (Avec les droits de lecture et d'écriture pour tout le monde).
/usr	**Arborescence qui contient les applications (*/usr/bin*) et leurs libraires (*/usr/lib*), ainsi que des commandes administrateur supplémentaires (*/usr/sbin*).**
/var	Données dynamiques qui sont écrites et lues pendant l'opération du système. Avec quelques sous-répertoires.

Fichiers spéciaux (*Device files*)

- Dans les systèmes Unix/Linux les périphériques sont accessibles à travers des fichiers spéciaux
  - Exemple: port série
    - Fichier spécial /dev/ttyS0
    - Écriture possible avec: echo 'Hello' > /dev/ttyS0
- Les fichiers spéciaux sont une interface simple pour communiquer avec le pilote d'un périphérique

Types de dispositifs

Dispositifs en mode caractère
- Sont capables de gérer des séquences d'octets d'une longueur arbitraire
- Écriture et lecture séquentielles
- Exemples :
  - Port série : écriture et lecture de messages arbitraires
  - Port parallèle : envoi de fichiers à l'imprimante

Dispositifs en mode bloc
- Travaillent avec des blocs de données de longueur fixe
- Permettent l'accès aléatoire aux données (chercher, sauter entre des blocs, ...)
- Exemple :
  - Une partition du disque dur

Gestion des fichiers spéciaux

- Les fichiers spéciaux se trouvent dans le répertoire /dev
- Un système Linux moderne peut contenir des milliers de fichiers spéciaux, pour tous les périphériques possibles

Méthodes de création des fichiers spéciaux

- Création manuelle :
  - Les fichiers spéciaux sont créés de manière statique lors de la configuration du système
- Création automatique :
  - Les fichiers spéciaux sont créés pendant le démarrage voire pendant l'opération du système en fonction des périphériques connectés
  - Deux systèmes disponibles pour Linux : **Devfs**, **uDev**

Création manuelle des fichiers spéciaux

- Solution préférable pour les systèmes embarqués
  - Les périphériques sont connectés de manière fixe et ne changent pas
  - Peu de périphériques à gérer
  - Solution très légère, sans utilitaire supplémentaire pour la gestion
- Pour des systèmes qui utilisent le port USB pour connecter des périphériques, une solution automatique est préférable

Fichiers spéciaux

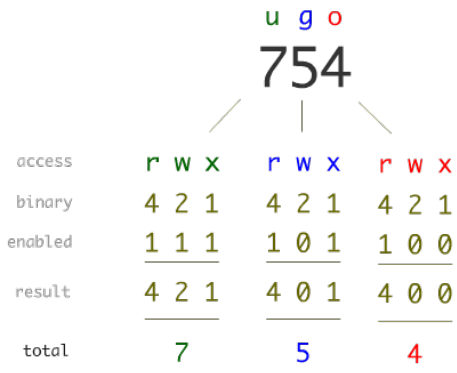
Un fichier spécial est défini par 4 paramètres

Nom	Permet l'accès par écriture/lecture
Mode	Caractère ou bloc
Numéro majeur	Indique le pilote responsable
Numéro mineur	Identifie le dispositif pour le pilote

Fichiers spéciaux indispensables

Fichier spécial	Type	Maj.	Min.	Perm.	Description
/dev/mem	char	1	1	600	Mémoire physique
/dev/null	char	1	3	666	Dispositif qui écarte toutes les données ('poubelle')
/dev/zero	char	1	5	666	Fournit des séquences d'octets 0x0
/dev/random	char	1	8	644	Générateur de nombres aléatoires
/dev/tty0	char	4	0	600	Console virtuelle actuelle
/dev/tty1	char	4	1	600	Première console virtuelle
/dev/ttyS0	char	4	64	600	Premier port série
/dev/tty	char	5	0	666	Terminal de contrôle du processus actuel
/dev/console	char	5	1	660	Console du système (accès par administrateur)
/dev/ram0	block	1	0	666	Ramdisk, si nécessaire

Permissions des fichiers linux



u : utilisateur
g : groupe
o : reste du monde (others)

r : lecture (read)
w : écriture (write)
x : e**X**écution

Création manuelle des fichiers spéciaux

- Avec la commande mknod
- Nécessite les droits d'administrateur

- Il faut aussi créer les liens symboliques pour les dispositifs virtuels :
  - stdin
  - stdout
  - stderr

```
> su  # Devenir administrateur
> cd $PROJROOT/rootfs/dev
> mknod -m 600 mem c 1 1
> mknod -m 666 null c 1 3
> mknod -m 666 zero c 1 5
> mknod -m 644 random c 1 8
> mknod -m 600 tty0 c 4 0
> mknod -m 600 tty1 c 4 1
> mknod -m 600 ttyS0 c 4 64
> mknod -m 666 tty c 5 0
> mknod -m 600 console c 5 1
> mknod -m 666 ram0 b 1 0      # Si nécessaire
> ln -s /proc/self/fd fd
> ln -s fd/0 stdin
> ln -s fd/1 stdout
> ln -s fd/2 stderr
> exit     # Redevenir l'utilisateur normal
```


Création automatique

Système Devfs

- Première solution pour la création et la destruction automatique de fichiers spéciaux
- Relation étroite avec le noyau
- Définit des noms fixes des fichiers spéciaux
  - /dev/lp0 : 1ère imprimante
  - /dev/lp1 : 2ème imprimante
- Abandonné à partir du noyau 2.6

Création automatique

Système udev

- Nouveau système, à partir du noyau 2.6
- Permet à l'utilisateur de définir des noms symboliques des périphériques en fonction des caractéristiques du périphérique
  - Constructeur, numéro de série, ...

- Fonctionnement
  - Lors de la connexion d'un périphérique, ses caractéristiques sont enregistrées dans une nouvelle arborescence /sys
  - L'utilisateur peut définir des noms symboliques dans le fichier /etc/udev/udev.rules
  - Exemple d'une règle dans udev.rules

```
BUS="usb", "SYSFS_serial="W0909020701241330" NAME=' lp_color'
```


Système de fichiers racine

- Arborescence des répertoires d'un système Linux
- Racine : répertoire '/'

Répertoire	Contenu
/bin	**Commandes utilisateur essentielles.**
/boot	Fichiers utilisés par le chargeur d'amorçage.
/dev	**Fichiers spéciaux qui représentent les périphériques du système, p.ex. /dev/usb.**
/etc	**Fichiers de configuration.**
/home	Répertoires des personnels des utilisateurs. Normalement pas présent sur un système embarqué.
/lib	**Librairies essentielles, comme la libraire C, et les modules du noyau.**
/mnt	Répertoire sous lequel on peut 'monter' d'autres systèmes de fichiers, comme par exemple le lecteur CDROM ou un lecteur disquette.
/proc	Système de fichiers spéciaux qui représentent des paramètres du noyau.
/root	Répertoire personnel de l'administrateur du système.
/sbin	**Commandes essentielles de l'administrateur.**
/tmp	Fichiers temporaires. (Avec les droits de lecture et d'écriture pour tout le monde).
/usr	**Arborescence qui contient les applications (/usr/bin) et leurs libraires (/usr/lib), ainsi que des commandes administrateur supplémentaires (/usr/sbin).**
/var	Données dynamiques qui sont écrites et lues pendant l'opération du système. Avec quelques sous-répertoires.

Commandes utilisateur Linux

- Un système Linux comprend des centaines de commandes utilisateur comme ls, vi, ifconfig, ftp, ...
- Elles se trouvent dans les répertoires /bin, /usr/bin, /sbin, /usr/sbin

La voie difficile

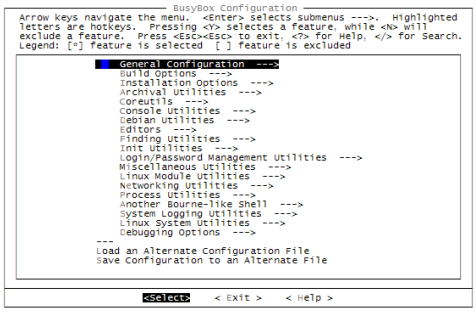
- Télécharger les différents paquetages comme *fileutils* (ls, cd, rm, ...), *net-tools* (ifconfig, route, ...), ...
- Les compiler manuellement
- Corriger les problèmes à cause de la compilation croisée

Busybox

- La voie simple !
- Paquetage qui combine plus de 200 commandes utilisateur
- Optimisé pour réduire la taille du système
  - N'implémente pas les options qui sont rarement utilisées
  - Combine toutes les commandes en un seul exécutable:
    /bin/busybox
  - Les différentes commandes ne sont que des liens symboliques vers l'exécutable busybox

Configuration de Busybox

- Outil de configuration similaire à menuconfig du noyau
- Lors de la configuration et de la compilation il faut indiquer :
  - Le préfixe du compilateur croisé
  - Le répertoire d'installation des exécutables




```
> make CROSS=armeb-xscale-linux-gnu- PREFIX=$PROJROOT/rootfs menuconfig
```


Compilation de Busybox

- Après la configuration

```
> make CROSS=armeb-xscale-linux-gnu- PREFIX=$PROJROOT/rootfs all
> make CROSS=armeb-xscale-linux-gnu- PREFIX=$PROJROOT/rootfs install
```

Résultat

- 110 commandes dans /bin, /sbin, ...
- Taille: 300 KB

```
> ls –l rootfs/bin
   7 Oct 17 16:53 ash -> busybox
296K Oct 17 16:53 busybox
   7 Oct 17 16:53 cat -> busybox
   7 Oct 17 16:53 chgrp -> busybox
   7 Oct 17 16:53 chmod -> busybox
   7 Oct 17 16:53 chown -> busybox
   7 Oct 17 16:53 cp -> busybox
   7 Oct 17 16:53 date -> busybox
   7 Oct 17 16:53 dd -> busybox
   ...
```

Système de fichiers racine

- Arborescence des répertoires d'un système Linux
- Racine : répertoire '/'

Répertoire	Contenu
/bin	**Commandes utilisateur essentielles.**
/boot	Fichiers utilisés par le chargeur d'amorçage.
/dev	**Fichiers spéciaux qui représentent les périphériques du système, p.ex. /dev/usb.**
/etc	**Fichiers de configuration.**
/home	Répertoires des personnels des utilisateurs. Normalement pas présent sur un système embarqué.
/lib	**Librairies essentielles, comme la libraire C, et les modules du noyau.**
/mnt	Répertoire sous lequel on peut 'monter' d'autres systèmes de fichiers, comme par exemple le lecteur CDROM ou un lecteur disquette.
/proc	Système de fichiers spéciaux qui représentent des paramètres du noyau.
/root	Répertoire personnel de l'administrateur du système.
/sbin	**Commandes essentielles de l'administrateur.**
/tmp	Fichiers temporaires. (Avec les droits de lecture et d'écriture pour tout le monde).
/usr	**Arborescence qui contient les applications (*/usr/bin*) et leurs libraires (*/usr/lib*), ainsi que des commandes administrateur supplémentaires (*/usr/sbin*).**
/var	Données dynamiques qui sont écrites et lues pendant l'opération du système. Avec quelques sous-répertoires.

Les fichiers de configuration

- Le répertoire /etc contient les fichiers de configuration du système
  - Configuration de base de Linux
    - Comptes des utilisateurs
    - Initialisation du système
    - Systèmes de fichiers à utiliser (partitions, répertoires spéciaux)
  - Configuration des applications
    - Comme par exemple interface graphique, carte de son, applications comme vi, emacs

Fichiers de configuration indispensables

Fichier ou répertoire	Contenu
/etc/inittab	Fichier de configuration de `init`, qui contrôle l'initialisation du système.
/etc/rc/ et **/etc/init.d/**	Répertoire avec les scripts de démarrage des différents services.
/etc/fstab	Définit les systèmes de fichiers à monter.
/etc/passwd	Contient les comptes des utilisateurs.
/etc/groups	Contient les groupes des utilisateurs.

Fichier /etc/passwd

- Contient une ligne pour chaque 'utilisateur' du système
  - On peut aussi définir des utilisateurs pour des services comme HTTP ou FTP avec des droits d'accès à des répertoires spécifiques
- Format de chaque ligne:

  <user>:<passwd>:<uid>:<gid>:<desc.>:<home>:<shell>
  - Exemples:
    root:xyYKMxfxkqMj2:0:0:root:/root:/bin/bash
    jehrensb:a3bEL45nnfCx:500:100:Juergen:/home/jehrensb:/bin/bash

Significations
- User : Nom d'utilisateur, comme p.ex. root ou labo
- Passwd : mot de passe crypté
- Uid : User ID numérique. 0 pour root, >500 pour les utilisateurs normaux
- Gid : Group ID numérique. 0 pour root, > 100 pour les utilisateurs normaux
- Desc. : Description optionnelle de l'utilisateur, p.ex. son nom complet
- Home : Répertoire personnel, p.ex. /home/labo
- Shell : Exécutable du shell utilisé par l'utilisateur, p.ex. /bin/bash ou /bin/tcsh

Création du mot de passe

- Le mot de passe de l'utilisateur doit être crypté dans le fichier passwd (MD5)
- Solutions
  1. Créer un utilisateur de test sur une machine normale et utiliser la commande passwd pour créer un mot de passe, puis le récupérer du fichier /etc/passwd
  2. Utiliser un petit programme C: *mkpasswd*

```
int main(int argc, char *argv[]) {
   char salt[255];
   strcpy(salt, argv[2]);
   printf("Key: %s  Salt: %s  Crypt: %s\n",
          argv[1],   salt,      crypt(argv[1], salt));
   return(0);
}
```

```
> ./mkpasswd rootroot xy
Key: rootroot  Salt: xy  Crypt: xyYKMxfxkqMj2
```


Fichier /etc/group

- Un utilisateur fait partie d'un ou plusieurs groupes
- Les droits d'accès aux fichiers et aux répertoires sont configurés pour le propriétaire et un groupe
- Le fichier /etc/group définit l'appartenance aux groupes
  - Contient une ligne par groupe
- Format de chaque ligne:

  <groupe>:<mot de passe>:<gid>:<liste des utilisateurs>
  - Exemple:
    users::100:labo,jehrensb

Significations
- Groupe : Nom du groupe
- Mot de passe : mot de passe crypté du groupe, normalement vide
- Gid : Group ID numérique. 0 pour root, > 100 pour les utilisateurs normaux
- List des utilisateurs : Nom d'utilisateurs, séparés par des virgules

Système de mots de passe cachés

- Problème du système classique
  - Le fichier /etc/passwd est lisible pour tout le monde
  - Attaque par dictionnaire est possible

Mots de passe cachés (*Shadow Password*)

  - Les mots de passe sont mis dans un fichier lisible par l'administrateur : /etc/shadow
  - Le fichier /etc/passwd reste lisible pour tous

Fichier /etc/passwd

```
root:x:0:0:root:/:/bin/sh
```

Fichier /etc/shadow

```
root:$1$abcdwxyz$Nbre0Q2.YHgHGw.m1SlD9/:::::::
```


Initialisation du système

- Après le démarrage du noyau, le processus '**init**' contrôle l'opération du système
  - Configure le système (p.ex. interfaces réseau)
  - Initialise les serveurs (Web, interface graphique, ...)
  - Crée des terminaux qui acceptent des log-ins

Fichiers de configuration de processus init

Fichier ou répertoire	Contenu
/etc/inittab	Fichier de configuration de `init`, qui contrôle l'initialisation du système
/etc/rc/ et **/etc/init.d/**	Répertoire avec les scripts de démarrage des différents services

Méthodes d'initialisation

- Les scripts d'initialisation varient beaucoup entre les différentes distributions
  - Les scripts sont différents
  - L'ordre de l'exécution des scripts est différent
  - L'organisation des répertoire est différente
  - Les outils de gestion sont différents
- Il y a deux systèmes principaux :

Initialisation Système V
  - Utilisée par la majorité des distribution Linux
  - Structure très claire des répertoires
  - Bien adaptée aux systèmes avec beaucoup de services

Initialisation BSD
  - Utilisée parfois sur les systèmes embarqués
  - Moins de scripts à gérer
  - Bien adaptée aux systèmes simples

Initialisation Système V

- Permet de définir plusieurs 'modes' (*runlevels*) qui influencent le comportement du système
  - Chaque runlevel correspond à un ensemble de services démarrés
  - On peut modifier le runlevel avec la commande telinit

Exemple d'une configuration de runlevels (RedHat)

Runlevel	Description
0	« *Halt* » : Le système est arrêté
1	« *Single-user mode* » : Un seul utilisateur (l'administrateur) peut se loguer. Mode maintenance.
2	Non utilisé
3	« *Full multi-user mode* » Plusieurs log-ins sont possibles, en mode ligne de commande.
4	Non utilisé
5	« *Graphical full multi-user mode* » Mode avec interface graphique X. Mode normal.
6	« *Reboot* » : Le système est redémarré.

Scripts d'initialisation SysV

- Organisation hiérarchique avec un répertoire par runlevel
  - Un script qui commence par « K » arrête un service
  - Un script qui commence par « S » démarre un service
  - Un numéro dans les noms des scripts indique l'ordre d'exécution
- Les fichiers dans /etc/rc?.d ne sont que des liens symboliques
- Les vrais scripts se trouvent dans /etc/init.d/
  - Acceptent les opérations « start », « stop », « restart »

```
/etc
├ rc0.d
├ rc1.d
├ rc2.d
├ rc3.d
├ rd4.d
├ rc5.d
├ init.d
```

Exemple : répertoire /etc/rc1.d/

- Script K20ssh
  - Lien vers /etc/init.d/ssh
  - Sera exécuté comme /etc/init.d/ssh *stop*
- Script S20mount
  - Lien vers /etc/init.d/mount
  - Sera exécuté comme /etc/init.d/mount *start*

Initialisation BSD

- Utilise le programme « init » de SysV
- Utilise le même format du fichier de configuration /etc/inittab
- Organisation différente des scripts d'initialisation
  - Un seul script par runlevel, p.ex. /etc/rc.d/rc.1
  - Le script doit arrêter et démarrer tous les services du runlevel

Système d'initialisation de Busybox

- Utilise un programme « init » simplifié
  - Ne permet pas de runlevels
  - Format du fichier /etc/inittab légèrement différent
- Exemple d'un fichier /etc/inittab

```
::sysinit:/etc/init.d/rcS
::askfirst:/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/sbin/swapoff -a
::shutdown:/bin/umount -a -r
::restart:/sbin/init
```

Décrit les opérationd à effectuer lors de l'initialisation, redémarrage, ...

Explication du fichier inittab

- Format de chaque ligne
  <id>:<runlevels>:<action>:<process>

  Exemple « ::sysinit:/etc/init.d/rcS »

  - **ID**: Peut indiquer le terminal qui sera utilisé par le processus (différent dans SysV)
  - **Runlevels**: vide, pour raisons de compatibilité
  - **Action**: indique quand et comment le processus est lancé
  - **Process**: commande à exécuter

Actions dans inittab

Action	Effet
sysinit	Premier action à exécuter. Le processus init attend la terminaison du processus lancé.
wait	Lancée après sysinit. Le processus init attend la terminaison du processus lancé avant de continuer avec d'autres opérations.
once	Lancée après les actions du type 'wait'. Le processus init n'attend pas la fin du processus lancé.
respawn	Lancée après les actions du type 'once'. L'action est exécutée et relancée chaque fois quelle termine.
askfirst	Similaire à l'action 'respawn' mais affiche le message « Please press Enter to activate this console. » et attend la touche Enter sur la console.
restart	Action à effectuer quand le processus init redémarre, par exemple à cause d'une erreur.
ctrlaltdel	Cette action est exécutée quand les touches Ctrl-Alt-Del sont préssées.
shutdown	Action à exécuter quand le système est instruit d'arrêter.

Scripts d'initialisation

- Un système embarqué typiquement n'a qu'un seul script d'initialisation

Script typique : ***$PROJROOT****/roots/etc/init.d/rcS*

```
# Mount /proc filesystem
mount /proc

# Remount rootfs, as indicated in /etc/fstab
mount –n –o remount,rw /

# Set hostname
/bin/hostname MAR2005

# Configure the network interfaces (called ixp, not eth)
# /sbin/ifconfig ixp0 192.168.1.1 netmask 255.255.255.0

# Démarrer le serveur SSH
# /sbin/sshd
```

Systèmes de fichiers

Système de fichiers (FS) :
partie de l'arborescence des répertoires

- L'arborescence complète se compose de plusieurs FS
- Les différents FS peuvent se trouver sur des médias de stockage différents
  - Partitions du disque dur, mémoire Flash, serveur réseau, mémoire RAM, ...
- Quelques FS sont virtuels
  - Créés dynamiquement pendant l'opération du système

Exemple

- Racine /

├ bin ├ etc ├ lib ├ usr ├ var ├ home	▪ Sur la partition /dev/hda1 ▪ Monté avec la commande mount /dev/hda1 /
├ labo ├ jehrensb	▪ Sur la partition /dev/hda2 mount /dev/hda2 /home
├ proc	▪ Système de fichiers virtuel mount –t proc none /proc

Le fichier /etc/fstab

- Définit les systèmes de fichiers à monter
- La commande mount -a dans le script d'initialisation monte tous les FS dans fstab

Exemple pour un système complet

```
#<file system> <mount point> <type>  <options>  <dump>  <pass>
none           /proc         proc    defaults   0       0
/dev/hda1      /             ext3    defaults   0       1
/dev/hda3      /home         ext3    defaults   0       2
/dev/hda2      none          swap    sw         0       0
/dev/hdd       /mnt/cdrom0   iso9660 user,noauto 0      0
```

Exemple pour un système embarqué

```
#<file system> <mount point> <type>  <options>  <dump>  <pass>
none           /proc         proc    defaults   0       0
/dev/ram0      /             ext2    defaults   0       1
```


Système de fichiers racine

- Arborescence des répertoires d'un système Linux
- Racine : répertoire '/'

Répertoire	Contenu
/bin	**Commandes utilisateur essentielles.**
/boot	Fichiers utilisés par le chargeur d'amorçage.
/dev	**Fichiers spéciaux qui représentent les périphériques du système, p.ex. /dev/usb.**
/etc	**Fichiers de configuration.**
/home	Répertoires des personnels des utilisateurs. Normalement pas présent sur un système embarqué.
/lib	**Librairies essentielles, comme la libraire C, et les modules du noyau.**
/mnt	Répertoire sous lequel on peut 'monter' d'autres systèmes de fichiers, comme par exemple le lecteur CDROM ou un lecteur disquette.
/proc	Système de fichiers spéciaux qui représentent des paramètres du noyau.
/root	Répertoire personnel de l'administrateur du système.
/sbin	**Commandes essentielles de l'administrateur.**
/tmp	Fichiers temporaires. (Avec les droits de lecture et d'écriture pour tout le monde).
/usr	**Arborescence qui contient les applications (/usr/bin) et leurs libraires (/usr/lib), ainsi que des commandes administrateur supplémentaires (/usr/sbin).**
/var	Données dynamiques qui sont écrites et lues pendant l'opération du système. Avec quelques sous-répertoires.

C'est fait !

Notre système Linux est complet

Répétition

Step 1: **Noyau**
- Configuration du noyau pour la plateforme cible
  - Architecture
  - Pilotes
  - Ligne de commande du noyau
- Compilation croisée

Step 2: **Librairies**
- Créés lors de la compilation de la toolchain
- Copier les librairies nécessaires dans le root FS
  - Enlever les librairies statiques
  - Enlever les librairies non nécessaires

Step 3: **Création des fichiers spéciaux**
- Création manuelle avec la commande mknod
  - Numéros majeurs et mineurs des différents dispositifs, type (char ou bloc)
- Création dynamique avec les systèmes DevFS ou uDev

Répétition cont.

Step 4: **Commandes utilisateurs**
- La voie simple: BusyBox
  - Fournit les 200 commandes les plus utilisés d'un système Linux
  - Réduit la consommation de mémoire
  - Problème de sécurité sur un système multi-utilisateurs
- La voie compliquée: télécharger et compiler les paquetages individuels

Step 5: **Comptes des utilisateurs**
- Fichiers /etc/passwd et /etc/groups
  - Création du mot de passe avec un petit programme
- Système de mots de passe cachés
  - Le fichier /etc/passwd ne contient plus les mots de passes cryptés
  - Le fichier /etc/shadow avec les mots de passe est lisible par l'administrateur

Répétition cont.

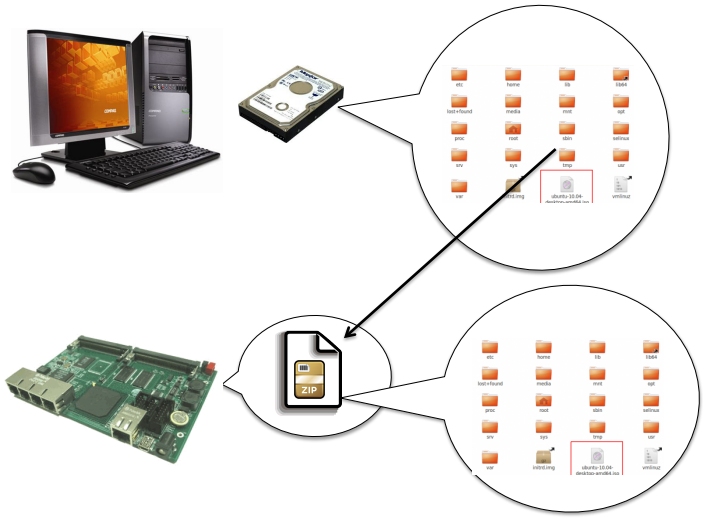
Step 6: **Initialisation du système**
- Processus init avec le fichier de configuration /etc/inittab
- Initialisation Système V
  - Inittab définit les scripts à exécuter à l'initialisation et dans chaque runlevel
  - Un répertoire de scripts pour chaque runlevel
- Initialisation BSD
  - Un seul script d'initialisation pour chacun des runlevels
- Initialisation BusyBox
  - Sans runlevels
  - Inittab définit les actions à exécuter lors de l'initialisation, pour le redémarrage, ...
  - Typiquement un seul script d'initialisation qui monte les FS, configure les interfaces réseau, démarre les service, ...

Step 7: **Configuration des systèmes de fichiers à monter**
- Systèmes de fichiers virtuels comme /proc
- Système de fichiers racine, typiquement un ramdisk

Au labo …

- Attention aux copy/paste
  - Ligne de commandes
  - Script
- Attention aux majuscules et minuscules
  - Fichiers spéciaux : par exemple tty**S**0
- /etc/passwd
  - Le mot de passe est chiffré !
  - mkpasswd.c

Image rootfs



Création de l'image d'installation

- Lors du démarrage, le root FS sera chargé en RAM et interprété comme un disque dur virtuel (ramdisk)
- On doit alors créer un fichier qui ressemble à une partition d'un disque dur

Procédure

1. Créer un fichier avec la taille requise (8 MB)
2. Formater le fichier comme une partition normale (p.ex. EXT2)
3. Monter la partition virtuelle
4. Copier le contenu de $PROJROOT/rootfs dedans
5. Démonter la partition virtuelle
6. Compresser le fichier

Création de l'image d'installation

Manuellement

```
> cd $PROJROOT/images
> dd if=/dev/zero of=ramdisk count=8192 bs=1k      # Créer un fichier vide
> mkfs -t ext2 -F ramdisk                          # Formater le fichier
> su                                               # Devenir root
> mount -o loop -t ext2 ramdisk /mnt               # Monter le fichier ramdisk sur mnt/
> cp –a $PROJROOT/rootfs/* /mnt/                   # Copier le rootfs dans ramdisk
> chown –R root:root /mnt/*                        # Changer le propriétaire des fichiers
> umount /mnt                                      # Démonter le ramdisk
> exit
> gzip -9 ramdisk                                  # Compresser le ramdisk
```

Avec l'utilitaire genext2fs

```
> cd $PROJROOT/images
> genext2fs -b 8192 -d $PROJROOT/rootfs/ ramdisk
> gzip -9 ramdisk
```

Amorçage de la cible

- Le chargeur d'amorçage ('*boot loader*') est un petit programme qui démarre l'OS de la cible
- Le chargeur d'amorçage est démarré par le BIOS
  - Sur une plateforme embarquée, il est normalement pré-installé dans la mémoire Flash ou dans un boot-ROM

Chargeurs d'amorçage des PC

- LILO
- GRUB
- NTLDR (Windows), BootX (Mac OS X)

Chargeurs d'amorçage des systèmes embarqués

- RedBoot
- U-Boot
- et beaucoup d'autres...

Configurations d'amorçage

Configuration de test

- Le noyau et le root FS sont téléchargés depuis un serveur
- Facilite le test du système
- Plusieurs possibilités
  - Téléchargement via TFTP
  - Montage d'un système de fichiers via le réseau

Configuration finale

- Le noyau et le root FS sont installés sur un dispositifs Flash ou un disque dur
- La cible est indépendant de serveurs et du réseau
- Utilisée sur l'équipement final

Amorçage sur réseau avec TFTP

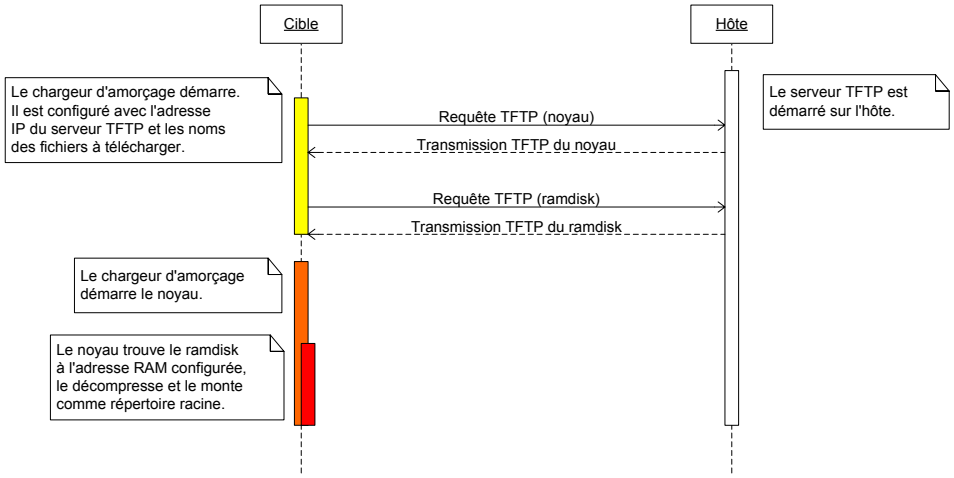
TFTP (*Trivial FTP*) :

- Protocole très simple pour le transfert de fichiers
- Permet de se connecter sur un serveur
- Opérations « *get* » et « *put* » pour télécharger un fichier

Séquence d'amorçage

1. Configurer la cible avec l'adresse IP du serveur TFTP
2. La cible télécharge le noyau et le place dans une zone de la mémoire RAM
3. La cible télécharge le ramdisk compressé et le place dans une zone de mémoire RAM
4. Le chargeur d'amorçage exécute le noyau
5. Le noyau trouve le ramdisk à l'adresse configurée, le décompresse et le monte comme le root FS

Amorçage sur réseau avec TFTP



Avantages et inconvénients TFTP

- Avantages
  - Très facile à configurer
  - Permet facilement de gérer plusieurs configurations différentes
- Inconvénients
  - Après chaque modification, un nouveau ramdisk doit être créé
  - Le transfert est lent

Amorçage avec TFTP et NFS

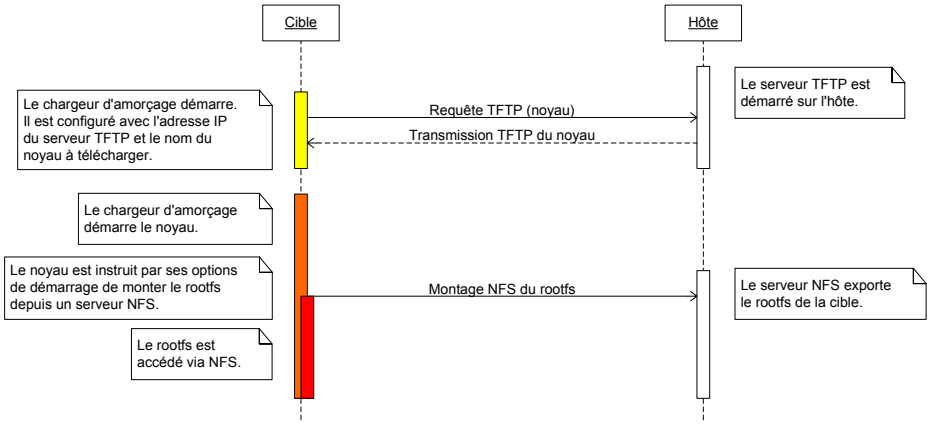
NFS (*Network File System*) :

- Permet de monter un répertoire qui se trouve sur un serveur
- Utilisable par le noyau pour monter le root FS à distance
- Avantages :
  - Il n'est plus nécessaire de créer un fichier ramdisk
  - Une modification du root FS sur l'hôte est directement visible sur la cible

Séquence d'amorçage

1. Configurer la cible avec l'adresse IP du serveur TFTP
2. La cible télécharge le noyau et le place dans une zone de la mémoire RAM
3. Le chargeur d'amorçage exécute le noyau
4. Le noyau monte le root FS via NFS

Amorçage avec TFTP et NFS



Amorçage depuis un dispositif Flash

- Configuration pour équipement finale, sans serveur
- Le noyau et le root FS sont installés sur le dispositif Flash
  - Le noyau n'est qu'un fichier qui est simplement copié dans une zone de la mémoire Flash
- Deux configurations possibles pour le root FS
  1. Ramdisk sur Flash
  2. Système de fichiers sur Flash

Ramdisk sur Flash

- Utilise le fichier compressé du ramdisk
- Ce fichier est copié dans la mémoire Flash
- Séquence de démarrage
  - Le noyau est chargé en RAM depuis le Flash
  - Le ramdisk est chargé en RAM depuis le Flash
  - Le noyau démarre, trouve le ramdisk dans la RAM et l'utilise comme root FS

Ramdisk sur Flash

- Avantages
  - Configuration simple
  - Utilise peu de mémoire Flash, comme le ramdisk est compressé
- Inconvénients
  - Ne permet pas d'enregistrer des modifications (p.ex. configuration)
    - Le ramdisk n'existe qu'en mémoire RAM

Système de fichiers sur Flash

- Stocker le root FS dans une ou plusieurs partitions sur le dispositif Flash

Avantages
- Demande moins de RAM, comme le root FS n'est pas chargé complètement en RAM
- Permet d'enregistrer des modifications

Inconvénients
- Demande plus de mémoire Flash, comme le root FS sera moins compressé
- Contraintes de la mémoire Flash
  - Quelques systèmes de fichiers sur Flash sont en lecture seule
  - Les opérations d'écriture abîment la mémoire Flash (10'000 – 1 mio d'opérations d'écriture par cellule)

Configuration typique avec plusieurs partitions Flash

Mémoire Flash

1. Partition des données statiques
   - Contient la majorité des répertoires du root FS
   - Formatée avec le format CRAMFS
     - Système en lecture seule
     - Bon taux de compression
2. Partition des données de configuration
   - Contient p.ex. le répertoire /etc
   - Formatée avec le format JFSS2
     - Format en lecture et écriture
     - Taux de compression plus faible
3. Fichier ramdisk avec les données dynamiques
   - Contient p.ex. le répertoire /var
   - Vit uniquement en mémoire
   - Évite d'abîmer la mémoire Flash
4. Noyau
   - Fichier copié dans la mémoire Flash

Chargeurs d'amorçage

Chargeur d'amorçage	Description	Moniteur	Architecture					
			x86	ARM	PPC	MIPS	SH	m68K
LILO	Le premier chargeur d'amorçage pour Linux sur x86, toujours très répandu.	Non	x					
GRUB	Le chargeur d'amorçage le plus utilisé sur x86.	Oui	x					
Etherboot	Intégré sur des cartes Ethernet récentes, permet télécharger le système depuis un serveur et de le démarrer en RAM.	Non	x					
Chargeur d'amorçage Compaq	Principalement utilisé sur iPAQ.	Oui		x				
U-Boot	Chargeur d'amorçage très flexible, Open Source.	Oui	x	x	x	x		x
RedBoot	Chargeur d'amorçage universel, de RedHat mais Open Source	Oui	x	x	x	x	x	x

RedBoot

- Chargeur d'amorçage très universel
- Open Source, de RedHat
- Fonction moniteur :
  ligne de commande interactive
- Permet l'amorçage avec TFTP/NFS et DHCP

```
+RedBoot(tm) bootstrap and debug environment [ROM]
Red Hat certified release, version 1.92p1 - built 16:53:03, Mar 26 2004

Platform: GW2342 Development Platform (XScale)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x04000000, 0x0001f880-0x03fd1000 available
FLASH: 0x50000000 - 0x51000000, 128 blocks of 0x00020000 bytes each.

Ctrl-C
RedBoot>
```

Quelques commandes de Redboot

Commande	Description
exec	Exécuter le noyau Linux.
fconfig	Modification de la configuration de RedBoot, notamment configuration du script de démarrage.
fis	Manipulation de la mémoire Flash.
help	Affiche de l'aide par rapport aux commandes de RedBoot.
ip_address	Configure l'adresse IP des interfaces.
load	Charge un fichier depuis le réseau, le port série ou un disque dur.
ping	Vérifie la connectivité et le fonctionnement des interfaces réseau.
reset	Réinitialise (redémarre) la plateforme.

Amorçage avec TFTP

```
RedBoot> ip_address –l 192.168.1.100 –h 192.168.1.1
IP: 192.168.1.100/255.255.255.0, Gateway: 192.168.1.1
Default server: 192.168.1.1, DNS server IP: 0.0.0.0

RedBoot> load –v –r –b 0x01600000 zImage
Using default protocol (TFTP)
Raw file loaded 0x01600000-0x01712e3f, assumed entry at 0x01600000

RedBoot> load -r -v -b 0x00800000 ramdisk.gz
Using default protocol (TFTP)
Raw file loaded 0x00800000-0x009799c2, assumed entry at 0x00800000

RedBoot> exec 0x01600000
Using base address 0x00800000 and length 0x001799c4
Uncompressing Linux.....................................................
..... done, booting the kernel.
Linux version 2.6.13.3 (jehrensb@edu-pc157.tcom.eivd.ch)
```

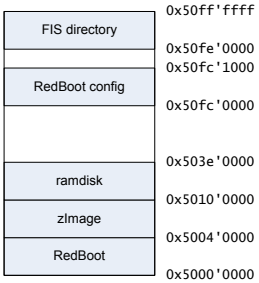
Création d'un script d'amorçage

- L'exécution des commandes d'amorçage peut être automatisée avec un script d'amorçage
- Commande **fconfig** pour configurer le script

```
RedBoot> fconfig
Run script at boot: true
Boot script:
Enter script, terminate with empty line
>> load -v -r -b 0x01600000 zImage
>> load -r -v -b 0x00800000 ramdisk.gz
>> exec 0x01600000
>>
Boot script timeout (1000ms resolution): 3
Use BOOTP for network configuration: false
Gateway IP address: 192.168.1.1
Local IP address: 192.168.1.100
Local IP address mask: 255.255.255.0
Default server IP address: 192.168.1.1
Console baud rate: 115200
DNS server IP address:
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Default network device: npe_eth0
Update RedBoot non-volatile configuration
- continue (y/n)? y
...
RedBoot>
```


Amorçage depuis le dispositif Flash

- Configuration la plus simple :
  Noyau et ramdisk sur le dispositif Flash
  - Ne nécessite pas de partitions sur le dispositif Flash
  - Au lieu de télécharger les images du noyau et du ramdisk, elles sont lues depuis le dispositif Flash



Mémoire Flash

Afficher le contenu du dispositif Flash

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x50000000  0x50000000  0x00040000  0x00000000
RedBoot config    0x50FC0000  0x50FC0000  0x00001000  0x00000000
FIS directory     0x50FE0000  0x50FE0000  0x00020000  0x00000000
zimage            0x50040000  0x01600000  0x000C0000  0x00800000
ramdisk           0x50100000  0x00800000  0x002E0000  0x00800000
```

Transfert des images sur le dispositif Flash

1. Effacer les anciennes images

```
RedBoot> fis delete zImage
Delete image 'zImage' (y/n)? y
... Erase from 0x50040000-0x50160000: .........
... Unlock from 0x50fe0000-0x51000000: .
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x03fdf000-0x03fff000 at ...
... Lock from 0x50fe0000-0x51000000: .
RedBoot> fis delete ramdisk
Delete image 'ramdisk ' (y/n)? y
... Erase from 0x50160000-0x502e0000:
... Unlock from 0x50fe0000-0x51000000: .
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x03fdf000-0x03fff000 at ...
... Lock from 0x50fe0000-0x51000000: .
RedBoot> fis list
Name              FLASH addr  ...
RedBoot           0x50000000  ...
RedBoot config    0x50FC0000  ...
FIS directory     0x50FE0000  ...
```

2. Créer les nouvelles images

```
RedBoot> load -v -r -b 0x01600000 zImage
Using default protocol (TFTP)
Raw file loaded 0x01600000-0x01712e3f, assumed ...
RedBoot> fis create zImage
... Erase from 0x50040000-0x50160000: .........
... Program from 0x01600000-0x01712e40 at 0x50040000:
... Unlock from 0x50fe0000-0x51000000: .
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x03fdf000-0x03fff000 at 0x50fe0000: .
... Lock from 0x50fe0000-0x51000000: .

RedBoot> load -v -r -b 0x00800000 ramdisk.gz
Using default protocol (TFTP)
Raw file loaded 0x00800000-0x009799c2, assumed ...
RedBoot> fis create ramdisk
... Erase from 0x50160000-0x502e0000: ............
... Program from 0x00800000-0x009799c3 at 0x50160000:
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x03fdf000-0x03fff000 at 0x50fe0000: .
... Lock from 0x50fe0000-0x51000000: .
```

3. Script de démarrage

```
fis load zImage
fis load ramdisk
exec 0x01600000
```