Systèmes d'exploitation pour l'embarqué – construction d'un Linux embarqué

Cours 5 - M2LSE
Jalil Boukhobza
Université de Bretagne Occidentale – Lab-STICC

Plan

Réduire l'empreinte mémoire !

- **Mémoire est très couteuse**
- 3 façons de **réduire l'empreinte mémoire**:
  1. **Optimiser le noyau**
     - Enlever le code dont on n'a pas besoin
     - Optimiser la compilation -O
     - Enlever le swap
     - Voir le « *Linux tiny kernel project* »: plusieurs optimisations sous forme de patch.
  2. **Optimiser l'espace de l'applicatif**
     - Optimiser son **code**
     - Optimiser son utilisation de la **librairie**
       - Optimiser les librairies partagées dans une application après développement
       - Utiliser des librairies réduites (uClibc, diet libc, etc)
     - Utiliser des **applications optimisées**:
       - *BusyBox*
       - *TinyLogin*
       - Serveur web *BOA*, *mini_httpd*, *GoAhead*
  3. **Compresser le système de fichiers**
     - Certains sont compressés: JFFS2, CRAMFS

Linux pour l'embarqué
Plusieurs solutions

Simplicité, rapidité

1. Solutions sur étagère
2. Construction d'une distribution (compilation croisée)
   1. Utiliser un(e) chaîne/framework déjà produit(e)
   2. Construire son environnement de compilation croisée

Complexité, flexibilité

Solutions sur étagère

- Dégrossir une distribution pour PC Fedora, Debian, etc.
  - Espace mémoire important
- Travaux communautaires:
  - **Ångström** www.angstrom-distribution.org: plusieurs bases de projets, génération en ligne d'image sur mesure (utilisation d'OpenEmbedded).
  - **Yocto project** www.yoctoproject.org : affilié à la Linux foundation (utilisation d'OpenEmbedded)
  - **Emdebian** www.emdebian.org : Embedded Debian Project fourni les outils de compil. Croisée + des distributions.
  - **Ubuntu**, **Rowaboat** (Android), **OpenWRT** (routeurs), **STLinux**, etc.
- Industriels: Windriver, Montavista, Bluecat …

Framework pour Linux embarqué

- Donne des possibilités proches du « *Do It Yourself* »
  - Outils de **construction de l'environnement** et du système: compilateur croisé, outils d'intégration, bibliothèque C, version du noyau, exécutables, etc.
  - Intégration des **patchs**
  - Résolutions des **dépendances**: compilées automatiquement
  - **Mises à jour**: intégration des dernières versions dans la chaîne

Framework pour Linux embarqué

- **Buildroot** http://buildroot.uclibc.org/ : ensemble de *makefile* et de patches permettant de générer:
  - Une chaîne de compilation croisée
  - Un système de fichiers racine
  - Une image de noyau Linux
- **OpenEmbedded** www.openembedded.org :
  - Utilisé dans Yocto et Ångström
  - Système de construction de distributions
    - **bitbake**: outil de construction dérivé de « portage » gestionnaire de paquet Gentoo
    - **OpenEmbedded**: fichiers de configuration, classes, recettes (décrivant les tâches à réaliser pour construire les paquets)
  - bitbake (grand chef cuisinier) s'appuie sur OpenEmbedded (livre de recettes) pour construire la liste des applications (le menu),
- Autre outils: **ELDK**, **uCLinux-dist**, **T2**, etc.

Compilation croisée

- Tout faire à la main → fastidieux et souvent inutile
- Utiliser un outil permettant d'aider à faire cela:
  - Crosstool-NG:
    - Outil de configuration simple (menuconfig)
    - Construction d'une chaîne de compilation croisée
- Différentes étapes:
  - Décider de la cible
  - Décider de la version de noyaux/gcc/glibc/binutils
  - Patches si nécessaire
  - Définir les préfixes (PREFIX, KERNEL_SOURCE_DIR, NATIVE, etc.)
  - Compiler binutils
  - Obtenir les (fichiers) entêtes du noyau pour la plateforme cible
  - Compiler un gcc minimal (pour la cible)
  - Construire la glibc (pour la cible)
  - Recompiler gcc complet (pour la cible)

Binutils

- Un ensemble d'outils permettant de générer et de manipuler des binaires pour une architecture donnée (CPU)
  - as, l'assembleur qui génère le code binaire d'un code source en assembleur
  - ld, l'éditeur des liens
  - ar, ranlib, génère l'archive .a, utilisée pour les librairies
  - objdump, readelf, size, nm, strings, pour inspecter les binaires
  - ...
- http://www.gnu.org/software/binutils/
- GPL licence

Binutils -2-

addr2line	Traduit les adresses de programme en noms de fichier et numéros de ligne ; suivant une adresse et le nom d'un exécutable, il utilise les informations de débogage disponibles dans l'exécutable pour déterminer le fichier source et le numéro de ligne associé à cette adresse
ar	Crée, modifie et extrait des archives
as	Un assembleur qui assemble la sortie de gcc en des fichiers objets
c++filt	Utilisé par l'éditeur de liens pour récupérer les symboles C++ et Java, et pour empêcher les fonctions surchargées d'arrêter brutalement le programme
gprof	Affiche les données de profilage d'appels dans un graphe
ld	Un éditeur de liens combinant un certain nombre d'objets et de fichiers archives en un seul fichier, en déplaçant leur données et en regroupant les références de symboles
nm	Liste les symboles disponibles dans un fichier objet
objcopy	Traduit un type de fichier objet en un autre
objdump	Affiche des informations sur le fichier objet donné, les options contrôlant les informations à afficher ; l'information affichée est surtout utile aux programmeurs qui travaillent sur les outils de compilation
ranlib	Génère un index du contenu d'une archive et le stocke dans l'archive ; l'index liste tous les symboles définis par les membres de l'archive qui sont des fichiers objet déplaçables
readelf	Affiche des informations sur les binaires du type ELF
size	Liste les tailles des sections et la taille totale pour les fichiers objets donnés
strings	Affiche, pour chaque fichier donné, la séquence de caractères affichables qui sont d'au moins la taille spécifiée (par défaut, 4) ; pour les fichiers objets, il affiche, par défaut, seulement les chaînes des sections d'initialisation et de chargement alors que pour les autres types de fichiers, il parcourt le fichier entier
strip	Supprime les symboles des fichiers objets
libiberty	Contient des routines utilisées par différents programmes GNU comme getopt, obstack, strerror, strtol et strtoul
libbfd	Bibliothèque des descripteurs de fichiers binaires (Binary File Descriptor)
libopcodes	Une bibliothèque de gestion des opcodes—la « version lisible » des instructions du processeur ; elle est utilisée pour construire des outils comme objdump

Source http://lfs.traduc.org/view/cfs-svn/x86_64-64/final-system/binutils.html

Etapes clés

1. Construction du noyau
2. Construction du système de fichiers racine
3. Construction des applications/commandes
4. Configuration du démarrage

1. Le noyau

1. **Sélection**
2. **Configuration**
   - Choix de ce qui va être inclut dans le **noyau**, ce qui ne va pas l'être, et ce qui va être inclut comme **module**.
   - Sélection de processeur
   - Sélection du support pour le matériel (bus, etc.)
   - Sélection des pilotes
   - Sélection de quelques options génériques du noyau
   - …
3. **Compilation**
   1. V ≤ 2.4.x (Construction des dépendances) - obsolète
   2. Nettoyage
   3. Compilation du noyau
4. **Installation** du noyau **ET** des modules

La sélection du noyau

- http://www.kernel.org/ : dépôt principal et officiel des (Vanilla ou mainstream) noyaux Linux.
- Pour l'embarqué … pas toujours **été** approprié! (question d'architecture), mais on utilise de plus en plus le dépôt principal.
- **Quelle version ?**: généralement prendre la dernière (stable) !
  - Mais ce n'est pas évident ! Cela dépend des changements qu'il y a eu
  - Les patchs (fonctionnalités additionnelles)
  - Compatibilité avec d'autres outils …

Architecture de processeur	Ou télécharger le noyau	Comment ?
x86	http://www.kernel.org/	ftp, http, rsync
ARM	http://www.arm.linux.org.uk/developer/ … plus très à jour (-> kernel.org)	ftp, rsync
PowerPC	http://penguinppc.org/	ftp, http, rsync, bitkeeper
MIPS	http://www.linux-mips.org/	cvs
SuperH	http://www.linux-sh.org/shwiki/FrontPage	cvs
M68k	http://www.linux-m68k.org/ … pas très à jour	ftp, http

Source: « Building Embedded Linux Systems » K.Yaghmour, O'Reilly 2003

Configuration du noyau

- Permet de sélectionner les options que l'on veut inclure dans le noyau (ou dans les modules)
  - Dépend de l'architecture sous jacente
- À la fin de la configuration, **sont générés**:
  - Un fichier .config
  - Un certain nombre de liens symboliques
  - Un certain nombre de fichiers entêtes
- Exemple d'options paramétrables :
  - Options de réseau
  - Support de modules chargeables
  - Dispositif de technologie mémoire
  - Périphériques bloc
  - Périphériques caractère
  - Support ATA/IDE/MFM/RLL
  - Support SCSI
  - Systèmes de fichiers
  - Son …

Configuration du noyau (2)

- Certaines options de la configuration dépendent de **l'architecture**, exemple:
  - Pas de port parallèle pour les powerpc (PPC)
  - Pas de port IEEE 1394 pour les MIPS
- Certains menus de la configuration sont spécifiques à certaines architectures.
- **Ce n'est pas parce qu'une option (n')est (pas) permise que le support pour l'implémenter (n') existe (pas)!**

Méthode de configuration

Dans le répertoire racine du source du noyau:

- `make config:` interface en ligne de commande avec demande pour chaque option (valeur par défaut si fichier .config existant)
- `make oldconfig:` prend en compte un *.config* existant et ne demande que les options non choisies dans l'ancien fichier.
- `make menuconfig:` affiche un menu basé sur un curseur (lib curse) en utilisant comme valeur par défaut un *.config* existant.
- `make xconfig:` affiche un menu (Xwindows) en utilisant comme valeur par défaut un *.config* existant.
- `make gconfig:` autre menu (pareil que *xconfig* mais utilisant d'autres librairies graphiques)
- Il est possible de sauvegarder plusieurs configurations
- Attention: certaines **libraires** (QT, GTK+, etc.) sont obligatoires pour lancer certaines des configuration ci-dessus

Mais encore …

Procédure de configuration:

- Chaque sous section du noyau définit les règles de configuration dans un **fichier séparé**: **Kconfig** du répertoire courant
- Dans ce fichier:
  - <nom=valeur> (dans le .config)
    - Nom: CONFIG_… (dans le .config)
    - Valeur: type (bool, tristate, string, integer, hexadecimal)
    - On peut spécifier s'il y a une valeur par défaut
  - Dépendances pour la visibilité de l'entrée
  - Un texte d'aide
- Configuration sauvegardée dans le fichier **.config** (racine du source), on y trouve:
  - *CONFIG_nomDeLaVariable=y* ou *m* ou autre
- Se retrouve dans le *Makefile* local sous la forme de

```
obj-$(CONFIG_nomDeLaVariable)+=fichierObjetDuPilote.o
```

- Se retrouve dans le *Makefile* global (directement ou indirectement) sous la forme de

```
obj-y(ou m)+= fichierObjetDuPilote.o
```

Voir http://www.linuxjournal.com/article/6568

Compilation du noyau

---obsolète---

Construction des dépendances (2.4.x):

- Plusieurs fichiers sources ⇨ plusieurs fichiers entêtes
- Dans chaque répertoire du source (version 2.4.x):
  - Un fichier *.depend* contenant les dépendances de chaque fichier de ce répertoire.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- clean dep
```

- ARCH=: architecture cible sur laquelle tournera le noyau (dans l'exemple: arm).
- CROSS_COMPILE=: besoin pour compiler le source. Utilisé pour la construction de nom d'outil utilisé pour la construction des noyaux (ex: nom du compilateur c est **arm-linux-gcc**)

Compilation du noyau (2)

Construction du noyau :

- Compilation du noyau avec:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- zImage
```

- ARCH=: architecture cible sur laquelle tournera le noyau (dans l'exemple: arm).
- CROSS_COMPILE=: besoin pour compiler le source. Utilisé pour la construction de nom d'outil utilisé pour la construction des noyaux (ex: nom du compilateur c est **arm-linux-gcc**)

- **zImage**: compressée avec gzip
  - Si vmlinux: image non compressée.
  - bzImage: question de taille et de format, rien à voir avec l'algo → du *gzip* dans les 2 cas (**non bzip**)
    - La différence ? Voir `Documentation/i386/boot.txt`

Compilation du noyau (3)

Construction des modules

- Une fois l'image du noyau proprement créée, on peut construire les modules:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux- modules
```

- (… dans le cas d'un arm bien sûr)
- … Il faut ensuite les installer
  - `$ make modules_install`

Mais au fait,… cela vaut-il le coup ?

2. Contenu du système de fichiers racine

- Quels répertoires y mettre ?
  - Plusieurs ne servent que dans le cas d'une utilisation multi-utilisateurs …
- Rappel (voir *File Hierarchy Standard* du TP1):

Répertoires	Contenu
bin	Binaires de commandes **essentielles**
boot	Fichiers statiques utilisés par le chargeur d'amorçage
dev	Périphériques et fichiers spéciaux
etc	Fichiers de configuration système incluant les fichiers de démarrage (initialisation)
home	Répertoire de travail des utilisateurs en plus d'entrées pour des services de type FTP.
lib	Librairies essentielles et modules
mnt	Point de montage pour les systèmes de fichiers temporaires
opt	Autre logiciels
proc	Système de fichiers virtuel pour les information du noyau et des processus
root	Répertoire de travail du root
sbin	Binaires des commandes **essentielles** pour l'administration système
tmp	Fichiers temporaires
usr	Hiérarchie secondaire contenant la plupart des applications et documents partagés par tous les utilisateurs. (serveur X)
var	Données dynamiques stockées par les applications

Contenu du système de fichiers racine (2)

- Les répertoires implémentant des fonctionnalités multiutilisateurs peuvent être omis:
  - `/home`, `/mnt`, `/opt` et (`/root` → `peut être inclus comme unique compte`)
- Quant au `/boot` …
  - Le chargeur d'amorçage peut-il extraire les images du noyau avant que le noyau ne soit démarré ?
- Créer certaines sous hiérarchies, ex: `/usr`, `/var`
- Bien réfléchir à ce qui est utile pour votre application embarquée et à ce qu'il l'est moins, ex: répertoire `/var/spool` inutile ! (voir la doc FHS pour être fixé)

Les librairies

- **glibc**: gourmande en mémoire …
- **uClibc:** http://www.uclibc.org/
  - vient du projet uClinux (linux sur des processeurs sans MMU).
  - Devenu un projet en soit: supporte les architectures avec ou sans MMU + support de plusieurs architectures
  - N'est pas basé sur GNU libc, ne suit pas les mêmes normes mais si ça compile sous Glibc, ça compilera *généralement* sous uClibC → C89, C99 et SUSv3 (standards)
  - LGPL
- **eglibc (Embedded Glibc www.eglibc.org)**:
  - Projet plus récent: rendre la glibc plus compatible avec l'embarqué
  - Plus petite empreinte mémoire
  - Configuration
  - Meilleur support de la compilation croisée
- **diet libc:** http://www.fefe.de/dietlibc/
  - Développé de 0 (ne découle pas d'un projet antérieur)
  - Ne supporte pas beaucoup d'architectures
  - Prévue pour être utilisée comme une librairie statique (contrairement à uClibc)
  - ATTENTION: GPL

Les librairies (2)

- Installation
- Configuration, exemple uClibc:
  - Options relatives à l'architecture cible
  - Configuration de librairie, options générales
  - Support réseau
  - Support chaînes de caractères
  - Options d'installation
  - ….
- Modification de l'environnement pour la prise en compte de la nouvelle librairie.
- **Aide**: Lire la doc et faire un essai indépendant du reste de la construction de votre Linux embarqué.

Les librairies (3)

Comment installer ces librairies dans le système de fichiers racine pour être utilisées à la volée ?

- On doit sélectionner quelles librairies inclure, exemple pour **glibc**:

librairie	Contenu	Commentaires
ld	Éditeur de liens dynamique	Obligatoire
libc	Librairie des fonctions C principales	Obligatoires.
libcrypt	Fonctions de cryptographie	Nécessaire pour les applications d'authentification
libm	Fonctions mathématiques	Nécessaire pour les fonctions mathématiques
libmemusage	Fonctions pour profiler les piles et le tas	Rarement utilisées
libpthread	Fonctions de threads Posix 1003.1c	Programmation multithreadée
librt	Fonctions d'E/S asynchrones.	Peu utilisées
:	:	:

- Déterminer les librairies dont on a besoin: commande ***ldd*** / application
- Copier celles dont on a besoin comme librairies + leurs liens symboliques

Les modules du noyau

- Ne pas oublier de copier les modules résultants de l'étape de compilation du noyau dans le système de fichiers racine … au bon endroit … (doc FHS!)
  - `/lib/modules` (pour une distribution classique)

L'image du noyau

- Si le noyau est démarré à partir du système de fichiers racine :
  - On copie l'image du noyau dans le répertoire **/boot** du système de fichiers racine.
  - On peut aussi y copier la configuration *.config*

Les fichiers périphériques

- Tous localisés dans /dev … tout est fichier!
- Voir: *Documentation/devices.txt* et faire un tour dans /dev
- Entrées basiques:

Nom de fichiers	Description	Type	Numéro majeur	Numéro mineur
mem	Accès à la mémoire physique	char	1	1
null	Périphérique null	char	1	3
zero	Source null	char	1	5
random	Générateur de nombres aléatoires	char	1	8
tty0	Console virtuelle courante	char	4	0
tty1	1ère console	char	4	1
tty	Console TTY courante	char	5	0
console	Console système	char	5	1

- Création avec **mknod** (heureusement usage de **udev**)
- Liens symboliques (ln –s) obligatoires: **fd, stdin, stdout, stderr**

…plus de détails lors d'un prochain cours !

Linux et le temps réel

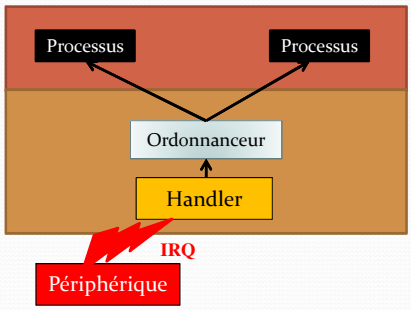
- Implémentation temps réel souple/mou de Linux: résolution du timer → quelques dizaine de µs, variabilité de quelques µs (dépend du matériel)
- Commutation de processus: quelques µs, entre threads: plus rapide
- Mémoire virtuelle: éviter les délais → ne pas swapper → verrouiller avec mlockall()
- Préemptibilité du noyau: plus fine depuis la version 2.6
- **Gestion des interruptions**: peut engendrer des délais importants

Patch Linux-rt (Preempt-rt)

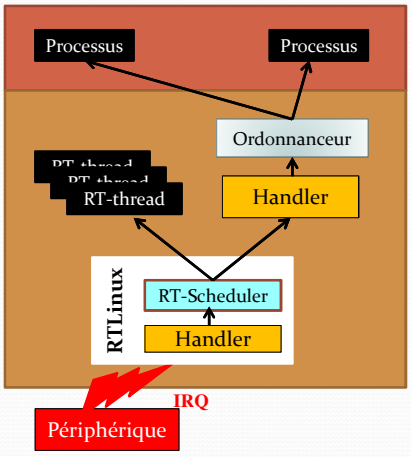
- Linux-rt (aussi appelé Preempt-rt): patch permettant d'améliorer les aspects temps réel:
  - Meilleure préemptibilité du noyau
  - Gestion des interruptions par threads (et pas par *tasklets* ou *workqueues* dans des thread noyau) → avoir une priorité moins importante que les tâches temps réel (mais plus de latence sur le traitement d'interruptions)

Extension temps réel de Linux

- Utilisation d'un ***nanokernel*** (RTLinux, RTAI, et Xenomai).
- Ajout d'un noyau prioritaire

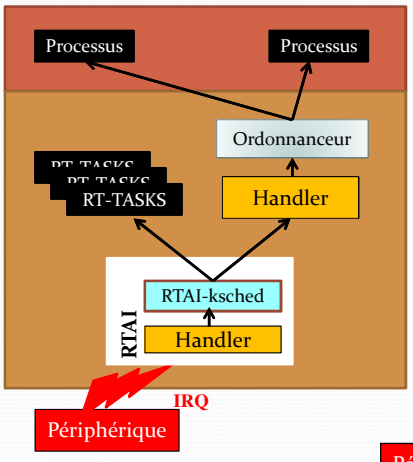


RTLinux

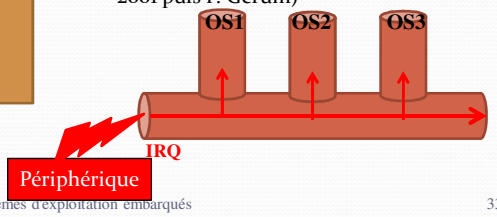


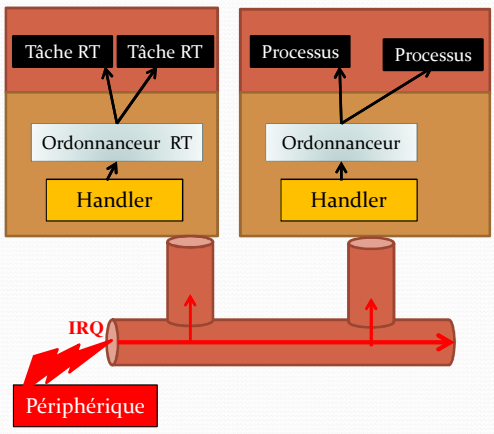
- Projet de Victor Yodaiken, Michael Barabanov ~1990
- Couche entre le contrôleur d'interruption et les gestionnaire du noyau Linux
- Ordonnanceur temps réel (priorité), tâches dans l'espace noyau
- RT-threads prioritaires par rapport aux tâches Linux
- Communications possible entre rt-thread et tâche conv. Au travers de FIFO, mémoires partagées, etc.
- 2 versions:
  - OpenRTLinux: versions simplifiée
  - Windriver Real-Time Core RTLinux: produit commercial, env. de développement + support technique,

RTAI et Adeos

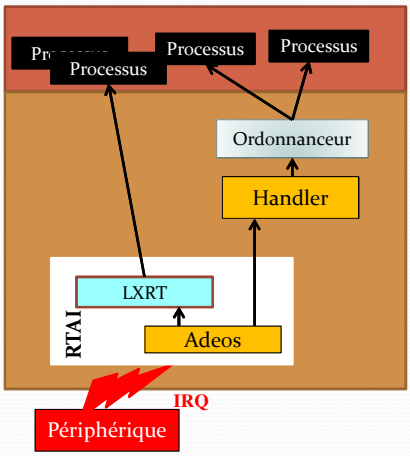


- Real Time Application Interface, même principe que RTLinux (developpé par l'équipe de P. Montegazza, EP Milan)
- Modèle similaire à RTLinux (pb après dépôt de brevet de ce dernier).
- Développement d'**ADEOS** (***Adaptive Domain Environnement for operating Systems***) : une couche logicielle basse capture les interruptions et les envoie dans un pipeline de systèmes d'exploitation (K. Yaghmour - 2001 puis P. Gérum)





ADEOS se greffe sur un noyau Linux

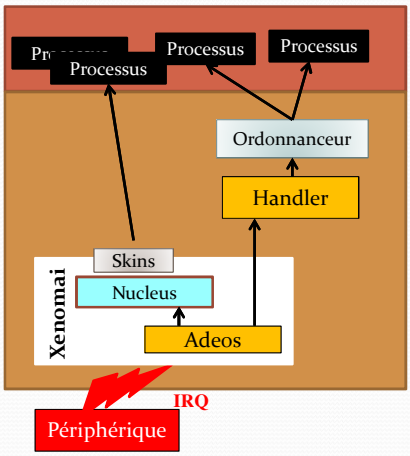


- 1ère versions RTAI: tâches temps réel → noyau (insertion de module)
- Evolution → tâches utilisateur gérées par LXRT
- Une tâche utilisateur peut se déclarer temps réel → appel à make_hard_realtime()

Xenomaï

- Crée par Philippe Gérum suite à des divergences de points de vue avec l'équipe originelle de RTAI (2005)
- Contrainte avec RTAI/LXRT: pas d'appel système sur un processus RT car sinon:
  - Noyau Linux reprend la main et peut faire autre chose … pb d'inversion de priorité
- Avec Xenomaï: autorisation des appels système
  - Événements autre que des interruptions peuvent circuler dans le pipeline Adeos (invocation d'appel sys., déclenchement scheduler…)
  - 2 modes d'exécution de thread rt:
    - **Mode primaire**: thread sous contrôle de Nucleus (ordonnanceur RT de Xenomaï)
    - **Mode secondaire**: thread ordonnancé par le noyau Linux

Suite (Xenomaï)



- Exécution initiale en mode primaire
  - → si appel systèmes → bascule en mode secondaire.
  - A la réception d'une interruption réveillant le processus → rebascule en mode primaire.
- Notion de « skin »: Xenomaï a une interface (API) prenant en compte plusieurs fonctionnalités: gestion de tâches, synchro, comm., → API native. Supports d'API d'autres RTOS/normes via ces « skins »
  - VXWorks, uITRON, PSOS, Posix ...

3. Applications principales

- Beaucoup (trop) de commandes dans les systèmes actuels:
- 2 possibilités:
  1. Ne prendre que les commandes dont on a besoin
     - Voir le projet **LinuxFromScratch** http://www.linuxfromscratch.org/
     - Sinon télécharger et (cross)compiler chaque application/package
  2. Dégrader/configurer un ensemble de commandes spécifiques à l'embarqué.
     - **BusyBox**: http://www.busybox.net/.
     - **TinyLogin** http://tinylogin.busybox.net/
     - **Embutils** http://www.fefe.de/embutils/

BusyBox

- Plusieurs commandes parmi lesquelles: *ar, cat, chgrp, chmod, chown, chroot, cp, cpio, date, dd, df, dmesg, dos2unix, du, echo, env, expr, find, grep, gunzip, gzip, halt, id, ifconfig, init, insmod, kill, killall, ln, ls, lsmod, md5sum, mkdir, mknod, modprobe, more, mount, mv, ping, ps, pwd, reboot, renice, rm, rmdir, rmmod, route, rpm2cpio, sed, stty, swapon, sync, syslogd, tail, tar, telnet, tftp, touch, traceroute, umount, uname, uuencode, vi, wc, which, et whoami.*
- Support architectural important
- Peut être liée statiquement ou dynamiquement à *glibc* et à *uClibc*
- On peut enlever certaines commandes
  1. Installation
  2. Configuration
  3. Compilation (pour l'archi cible) et copie/installation

Lire la doc !

TinyLogin (fusionné avec BusyBox)

- Utilitaires de **login** en un seul binaire
- Mêmes développeurs que *BusyBox* (facile de les faire fonctionner ensemble)
- Remplace les commandes: *addgroup, adduser, delgroup, deluser, getty, login, passwd, su, sulogin*, et *vlock.*

Embutils

- Par les développeurs de *Diet libc*
- Support pour l'ARM, i386, PPC, et MIPS
- Un binaire pour toutes les commandes et un petit binaire pour chacune: *arch, basename, cat, chmgrp, chmod, chown, chroot, chvt, clear, cp, dd, df, dirname, dmesg, domainname, du, echo, env, false, head, hostname, id, install, kill, ln, ls, md5sum, mesg, mkdir, mkfifo, mknod, mv, pwd, rm, rmdir, sleep, sleep2, soscp, sosln, soslns, sosmv, sosrm, sync, tail, tar, tee, touch, tr, true, tty, uname, uniq, wc, which, whoami, write*, et *yes*
- Ne peut être que **statiquement** lié à Dietlibc
  1. Installation
  2. Configuration
  3. Installation sur l'archi cible

4. Initialisation du système

- Dernière action de l'initialisation du noyau: **init**
  - Finalise l'initialisation en lançant des applications clés
- Peut être même remplacée par une application en configurant l'amorçage
  - L'application sera la seule à être exécutée.
- http://www.linux.it/~rubini/docs/init/
- Plusieurs possibilités:
  - Init **System V** standard
  - Init de la **BusyBox**
  - **Minit** (miniature init) de Dietlibc http://www.fefe.de/minit

Init System V (Linux)

- https://wiki.archlinux.org/index.php/SysVinit
- Installer
- Une fois installé, il faut:
  - Ajouter le fichier ***etc/inittab*** approprié (*event.d* pour Ubuntu, *upstart*): définit les niveaux d'exécution (*run levels*) du système … **à voir** !
  - Remplir le répertoire **etc/rc.d** avec les fichiers appropriés: fichiers définissant ce qu'il y a à faire pour chaque niveau d'exécution.
- **7 niveaux d'exécution** pour les *init systemV*

Niveau d'exécution	Description
0	Le système est arrêté
1	Mono utilisateur, pas besoin de login, mode texte
2	Mode multiutilisateur sans NFS, login en ligne de commande
3	Mode multiutilisateur complet, login en ligne de commande, mode texte
4	Non utilisé
5	Login en mode graphique, multiutilisateur complet, X11
6	Redémarrage du système

Init System V (2)

- Chaque niveau d'exécution correspond à un ensemble d'applications/services
- Ex: niveau d'exécution 5 (niveau par défaut), X11 est lancé pour demander un login et mot de passe
- Lorsqu'on passe d'un niveau à un autre, les services du premier sont arrêtés et ceux du second sont démarrés.
- Pour les systèmes embarqués, par défaut, le **niveau 1** peut être suffisant (si pas besoin d'identification ou d'interface graphique)
- … Création de */dev/initctl* (fifo) permettant de communiquer entre différents niveaux d'exécution lors d'un changement de ces derniers.

Init de BusyBox

- Ne procure pas de support pour les niveaux d'exécution
- Plus simple et léger, et programmé explicitement pour les systèmes embarqués
- *Init* de la *BusyBox*:
  1. Installe les *handlers* des signaux **Voir la doc !**
  2. Initialise les consoles
  3. Parse le fichier etc/inittab
  4. Exécute le script d'initialisation du système */etc/init.d/rcS*
  5. Exécute toutes les commandes bloquantes d'inittab (type d'action: wait).
  6. Exécute toutes les commandes qui ne s'exécutent qu'une seule fois (type d'action: once).
  7. Une fois cela effectué:
     - Exécute toutes les commandes *inittab* à régénérer (avec ou sans intervention de l'utilisateur)

5. Installation du système de fichiers racine

- ~4 possibilités:
  1. Système de fichiers sur RAM
  2. Système de fichiers en réseau
  3. Système de fichiers sur flash
  4. Système de fichiers sur disque
- Plusieurs système de fichiers ?
- Paramètres pour caractériser un système de fichiers:
  - **Écriture**
  - **Persistance (/volatilité)**
  - **Recouvrement** d'une panne d'alimentation
  - **Compression**
  - Support: dans la **RAM ?**

Choix du système de fichiers

Système de fichiers	Ecriture	Persistance	Recouvrement d'une panne d'alimentation	Compression	Dans la RAM
CRAMFS	Non	N/A	N/A	Oui	Oui
JFFS2	Oui	Oui	Oui	Oui	Non
Ext2 sur mem flash	Oui	Oui	Non	Non	Non
Ext3 sur mem flash	Oui	Oui	Oui	Non	Non
Ext2 sur RAM disk	Oui	Non	Non	Non	Oui
:	:	:	:	:	:

Source: « Building Embedded Linux Systems » K.Yaghmour, O'Reilly 2003

Système de fichiers disque sur RAM (RAMDISK)

- Vie dans la RAM et agit comme un **périphérique bloc**
- Les *Ramdisk* sont remplies en utilisant des images compressées de systèmes de fichiers disque (ex: ext2)
  - Initialisation du système … **initrd**
- Au démarrage, le noyau vérifie la présence d'un *initrd*
  - Extraction de l'image du système de fichiers (compressée ou pas) d'un média de stockage vers la ramdisk …
- ext2 plus souvent utilisé pour *l'initrd* …. (plutôt initramfs actuellement)
- Il faut d'abord créer *l'initrd* que l'on va charger au démarrage sur le disque …

Exemple

- **Création d'une image de 8Mo initialisée avec des zéros:**

```
$ cd ${PRJROOT}
$ mkdir tmp/initrd
$ dd if=/dev/zero of=images/initrd.img bs=1k count=8192
 8192+0 records in
 8192+0 records out
```

- **Création d'un sys de fichiers dans un fichier (-F) sans blocs spécifiques au super utilisateur (-mo)**

```
# /sbin/mke2fs -F -v -mo images/initrd.img
```

mke2fs 1.18, 11-Nov-1999 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
2048 inodes, 8192 blocks
:

- **Rendre le fichier (contenant le système de fichiers) « montable » en lui associant un périphérique boucle (loop) – un fichier visible comme un périphérique de type bloc**

```
# mount -o loop images/initrd.img tmp/initrd
```


Exemple (2)

- On peut copier le système de fichiers entier dans *l'initrd*:

```
# cp -av rootfs/* tmp/initrd
rootfs/bin -> tmp/initrd/bin
rootfs/bin/busybox -> tmp/initrd/bin/busybox
rootfs/bin/ash -> tmp/initrd/bin/ash
rootfs/bin/cat -> tmp/initrd/bin/cat
rootfs/bin/chgrp -> tmp/initrd/bin/chgrp...
# umount tmp/initrd
```

- Le fichier ***images/initrd.img*** contient à présent une image de notre système de fichiers racine entier
- Pour le compresser:

```
$ gzip -9 < images/initrd.img >
  images/initrd.bin
```

6. Paramètre de **démarrage**

- Vocabulaire:
  - Bootloader (chargeur de démarrage/amorçage): ne s'occupe « que » du démarrage.
  - Moniteur : démarrage + interface en ligne de commande pour le débogage.

			Architectures				
Bootloaders	**Moniteurs**	**Description**	x86	ARM	PowerPC	MIPS	m68k
LILO	Non	Chargeur principal pour Linux	X				
GRUB	Non	Successeur GNU de LILO	X				
ROLO	Non	Charge Linux d'une ROM sans BIOS	X				
Loadlin	Non	Charge Linux de DOS	X				
Etherboot	Non	Chargeur (mis en ROM) permettant de démarrer à travers une carte réseau	X				
LinuxBIOS	Non	Remplacement de BIOS basé sur Lniux	X				
sh-boot	Non	Chargeur prncipal pour le projet LinuxSH					
U-Boot	Oui	Chargeur universel basé sur PPCBoot et ARMBoot	X	X	X	X	
RedBoot	Oui	Chargeur basé sur eCos (RedHat)	X	X	X	X	X

Source: « Building Embedded Linux Systems » K.Yaghmour, O'Reilly 2003

Chargeurs de démarrage

- LILO: http://www.linux-france.org/article/sys/chargeurs/ix86/lilo/boot-lilo.html
- GRUB: http://www.gnu.org/software/grub/
- ROLO (ROmable LOader): charge Linux directement de la ROM sans BIOS ftp://ftp.elinos.com/pub/elinos/rolo/
- Etherboot:
  - Plusieurs cartes réseaux permettent l'insertion d'une ROM (exécutée au démarrage)
  - http://etherboot.sourceforge.net/
- LinuxBIOS: remplacement complet du BIOS http://www.linuxbios.org/.
- **U-Boot: http://sourceforge.net/projects/u-boot plus riche/flexible/à jour. Supporte plusieurs archis, peut démarrer d'un disque IDE et SCSI (doc: http://www.denx.de/wiki/U-Boot)**
- RedBoot: http://sources.redhat.com/redboot/ à l'origine destiné pour eCos. Il a un support conséquent pour les différentes architectures

U-Boot

- *Universal bootloader:* support de plusieurs architectures (ARM, PowerPC, MIPS, SH4, etc.)
- Multi plateformes et Open Source
- Syntaxe similaire à l'environnement Unix (bash)
- Caractéristiques (http://www.stlinux.com/u-boot) :
  - Téléchargement réseau: tftp, dhcp, nfs, bootp
  - Téléchargement série: binaires (via Kermit)
  - Gestion de mémoire flash: copie, protection, cramfs, jffs2
  - Types de flash: CFI NOR, NAND
  - Utilitaire mémoire: copy, dump, crc, check, mtest
  - Stockage de masse: IDE, SATA, USB
  - Démarrage du disque: blocs brutes, ext2, fat, reiserfs
  - Shell interactif

Commandes de U-Boot

- Utilisation de variables d'environnement (*setenv*, *printenv*), enregistrement avec *saveenv* dans une zone de mémoire flash réservée.
- `get` et `set` de variables d'environnement
- Exécution de macro-instructions (ensemble d'instructions séparées par des « ; »)
- Lecture/écriture de la mémoire
- Configuration du réseau
- Exécution de binaires au format SREC (S-Record de Motorola) ou de noyau au format uimage
  - uimage est le format zimage classique + entête U-Boot

Quelques commandes

- `printenv`, `setenv [var name] [var value]`, `saveenv`
- `run [var name]`: exécution de macro
- Commandes flash NOR:
  - `erase`: efface une zone flash NOR
  - `cp [.b, .w, .l] [source] [target] [count}`
  - …
- Commandes flash NAND:
  - `nand erase [addr] [len]`: efface la flash
  - `nand write [source] [dest] [count]` écrit les données de la RAM à la flash
  - `nand read [dest] [source] [count]`
- `boot`: exécution de la macro bootcmd
- `bootm` : démarrage à une @ RAM
- `bdinfo` affiche les info de la carte
- `flinfo`: affiche les info de la flash
- `dhcp`; @ dhcp puis chargement de ${bootfile}
- `tftpboot`: chargement et démarrage d'un fichier en RAM via tftp
- `help`: affichage de l'aide
- …