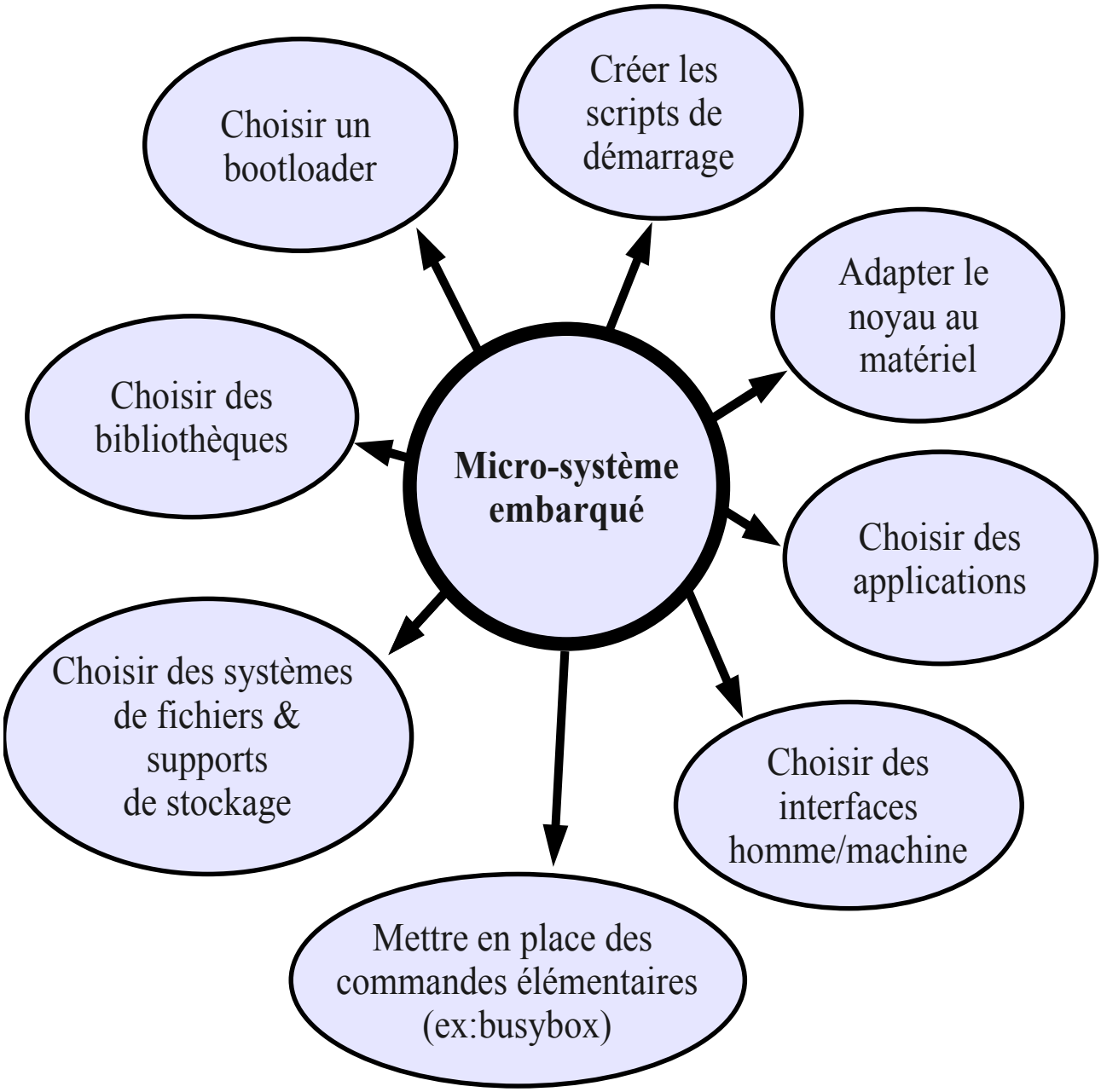
Méthodologie de création d'un micro-système embarqué.

1. Étapes de création d'un micro-système.



2. Stratégies de création d'un système embarqué.

La stratégie de création va dépendre de l'architecture et du type et du nombre d'applications voulues :

- ***adapter une distribution existante*** : si on a besoin d'un ensemble d'applications comme pour une borne internet par exemple. Cas particulier, un périphérique qui n'est supporté que sur une distribution bien précise. Cette méthode ne permet pas d'optimiser au maximum le système embarqué.
- ***Créer un système embarqué linux de A à Z*** : si le nombre d'application est faible ou si elle sont « faites maison ». Cette méthode permet d'avoir des systèmes extrêmement optimiser : seul les éléments nécessaires sont mis en oeuvre. Ex : baladeur mp3, routeur ...
- ***Prendre les applications (et les bibliothèques) d'une distribution et construire le reste*** (noyau, scripts de démarrage ...). Cette solution intermédiaire s 'apparente un peu « à du bricolage » mais peut être pratique dans certains cas.

3. Les contraintes d'un système embarqué.

Suivant la destination du système embarqué, on cherchera à optimiser suivant les axes suivants :

- l'espace occupé sur le système de fichier,
- l'empreinte mémoire du système,
- la rapidité d'exécution (et/ou d'accès au système de fichier),
- la rapidité de démarrage,
- la stabilité suivant les manœuvres d'un utilisateur final (ex : arrêt brusque du système par coupure de l'alimentation),
- ....

Choix de GNU/Linux pour un système embarqué.

Préambule : l'ensemble de ce cours est fortement inspiré de documents provenant de http://www.free-electrons.com. Ne pas hésiter à consulter ce site !

1. Signification du terme GNU/Linux.

GNU = Gnu is not Unix (acronyme non résolu)
Linux = noyau (kernel) initié par Linus Torvalds en 1991

Ainsi *GNU/Linux* = noyau + outils = système d'exploitation

2. Les licences.

Les logiciels libres :
« Nous maintenons cette définition du logiciel libre pour décrire clairement les conditions à remplir pour qu'un logiciel soit considéré comme libre.

L'expression «Logiciel libre» fait référence à la liberté et non pas au prix. Pour comprendre le concept, vous devez penser à la «liberté d'expression», pas à «l'entrée libre».

L'expression «Logiciel libre» fait référence à la liberté pour les utilisateurs d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel. Plus précisément, elle fait référence à quatre types de liberté pour l'utilisateur du logiciel :

- *La liberté d'exécuter le programme, pour tous les usages (liberté 0).*
- *La liberté d'étudier le fonctionnement du programme, et de l'adapter à vos besoins (liberté 1). Pour ceci l'accès au code source est une condition requise.*
- *La liberté de redistribuer des copies, donc d'aider votre voisin, (liberté 2).*
- *La liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté (liberté 3). Pour ceci l'accès au code source est une condition requise. »*[1]

Le système GNU/Linux est un ensemble de logiciels libres. Ainsi, développer un système embarqué permet de réutiliser du code produit par d'autres (« ne pas réinventer la roue ») comme les couches réseaux ...
Exemples de Licences : GPL, LGPL, BSD ... [2]

3. La portabilité.

GNU/Linux a été porté sur différentes architecture :

- 32 bits : X86, PowerPC, alpha, arm ...
- 64 bits : ia64, ppc64 ...

La liste des architectures se trouve dans le noyau : répertoire arch/
De plus, par la cross-compilation, on peut développer sur une architecture différente de l'architecture de la cible.

4. L'adaptabilité (« scalability »).

Les systèmes GNU/Linux peuvent être adaptés suivant leur utilisation : du système le plus réduit au supercalculateur, du routeur au cluster en basant par les terminaux graphiques, les postes de bureau, serveurs web, serveurs de fichier, baladeurs MP3, CD-ROM bootable, clé USB ...

5. Le support réseau.

GNU/Linux offre un support réseau complet : différents protocoles (SMB, IPX, NFS...) et répond aux normes de l'IETF[3] (TCP/IP v4, et v6, DHCP ...).

6. La modularité.

L'architecture même du système offre une grande modularité ; par exemple, il existe plusieurs couches

---

1 *Qu'est ce qu'un logiciel libre ? http://www.gnu.org/philosophy/free-sw.fr.html*
2 *Pour en savoir plus : http://www.gnu.org/licenses/license-list.fr.html*
3 *http://www.ietf.org/home.html*

graphiques possibles (X11, framebuffer ...) et on n'est pas obligé d'installer la couche graphique si on n'en a pas besoin !!
Enfin, du fait des licences employées, on construit un système linux à partir d'éléments déjà existants : noyau, driver, drivers, bibliothèques ...

7. Les distributions linux.

Il existe des distribution commerciales (soutenues par une entreprise) et des distributions communautaires.

- ***Mandriva*** : http://www.mandriva.com
- ***RedHat*** : http://redhat.com
- ***YellowDog*** : http://terrasoftsolutions.com
- ***Debian*** : http://www.debian.org
- ***Gentoo*** : http://www.gentoo.org
- ***Ubuntu*** : http://www.ubuntu-fr.org/
- ....[4]

Le choix d'une distribution se fait à partir des caractéristiques de cette dernière et des objectifs qu'elle se fixe : serveur, poste de travail, ...

Il existe aussi des distributions spécialisée dans l'embarqué :

- ***uCLinux*** (pour micro-contrôleurs, sans MMU) : http://www.uclinux.org/
- ***Emdebian*** (debian pour systèmes embarqué, en cours de développement) http://www.emdebian.org/
- ***PeeWeelinux*** : distribution pour systèmes embarqués. http://www.peeweelinux.org
- ...

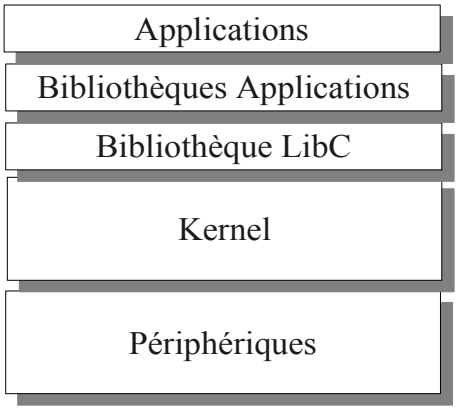
8. Le choix de GNU/Linux d'un point de vue industriel.

- limitation des coûts de Licence (cela n'est qu'une conséquence, mais n'est pas un but des licences libres : « *you should think of ``**free**'' as in ``**free speech**,'' not as in ``**free** beer.* »[5]),
- accès facile à la documentation : tout le monde peut lire les codes sources existants,
- limitation du coût des outils de développements et du matériel (PC + carte cible standards),
- indépendance technologique,
- stabilité du système,
- compatibilité avec les standards et normes,
- un grand nombre d'applications disponibles,
- ....

4 *Voir le site Distrowatch pour une liste complète : http://distrowatch.com*
5 *www.gnu.org/philosophy/**free**-sw.html*

Structure d'un système GNU/Linux

1. Structure générale du système GNU/Linux.



On a une architecture par couches.

2. L'arborescence des fichiers.[6]

- entièrement hiérarchique (avec *un unique* 'arbre des répertoires'),
- déterminé par la *fonction* d'un fichier et non par le programme qui lui sert de contexte (pas de 'répertoire de programme'),
- sous GNU/Linux, tout n'est que fichiers (ou processus) . Ainsi, les périphériques sont vus comme des fichiers.

/root

/bin	binaires de commandes utilisateurs essentielles (pour tous les utilisateurs)
/boot	fichiers statiques du chargeur de démarrage
/dev	Fichiers de périphériques
/etc	Configuration spécifique à la machine
/home	Répertoires personnels (optionnel)
/lib	Bibliothèques de bases du système et modules noyau
/media	Points de montages pour les systèmes de fichiers montés temporairement.
/mnt	Points de montages pour les systèmes de fichiers montés
/opt	Paquetages de logiciels d'applications supplémentaires
/root	Répertoire utilisateur root (optionnel)

6 *http://www.linux-france.org/article/sys/fhs/fhs-3.html*

|_____ /sbin Binaires systèmes (pour root et consort)

|_____ /tmp Fichiers temporaires

|_____ /usr Hiérarchie secondaire

|_____ /var Données variables

Exemples de systèmes de fichiers virtuels :

|_____ /proc Systèmes de fichiers virtuels donnant les information sur le noyau et le système (processus, drivers, configuration noyau ..)

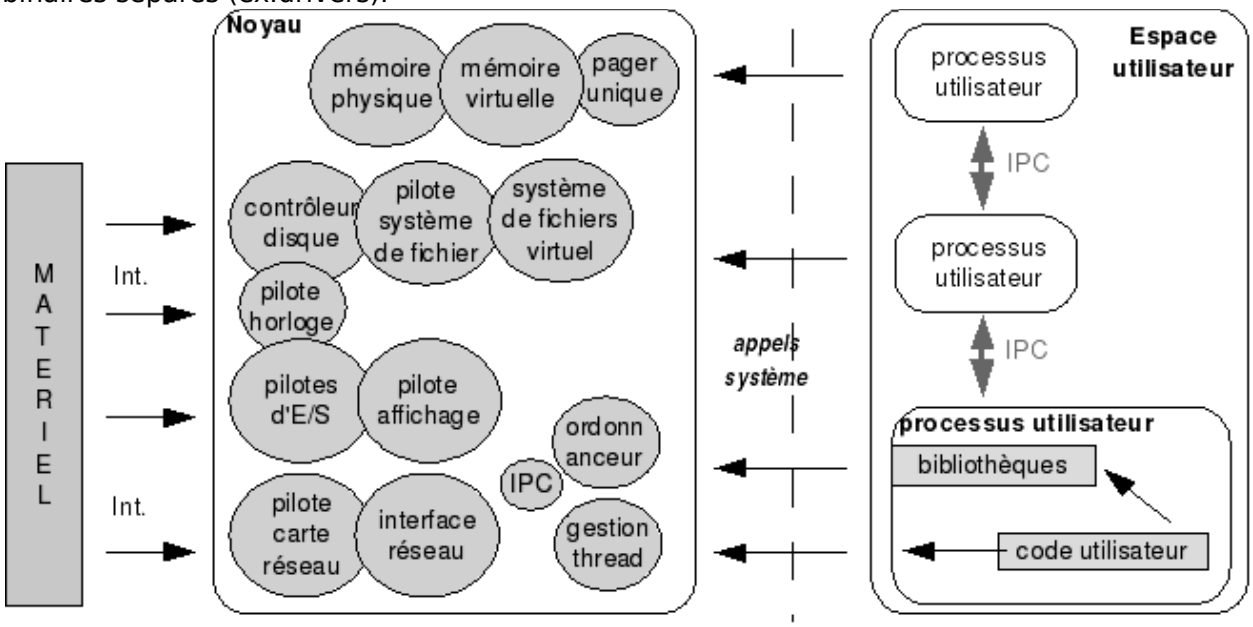
Montage :

|_____ /sys mount -t proc none /proc

mount -t /sys none /sys

3. Architecture du noyau.

Le noyau Linux est un noyau de type ***monolithique*** (l'ensemble des fonctions sont regroupées dans un seul bloc de code).
Afin de simplifier la maintenance du code, le noyau linux est devenu ***modulaire*** : les fonctions principales du système sont dans un bloc unique et les autres fonctions sont dans des blocs de code et binaires séparés (ex:drivers).



Architecture d'un noyau monolithique[7]

Remarque : il existe aussi des micro-noyaux où un maximum de fonctionnalités sont placées dans l'espace utilisateur.

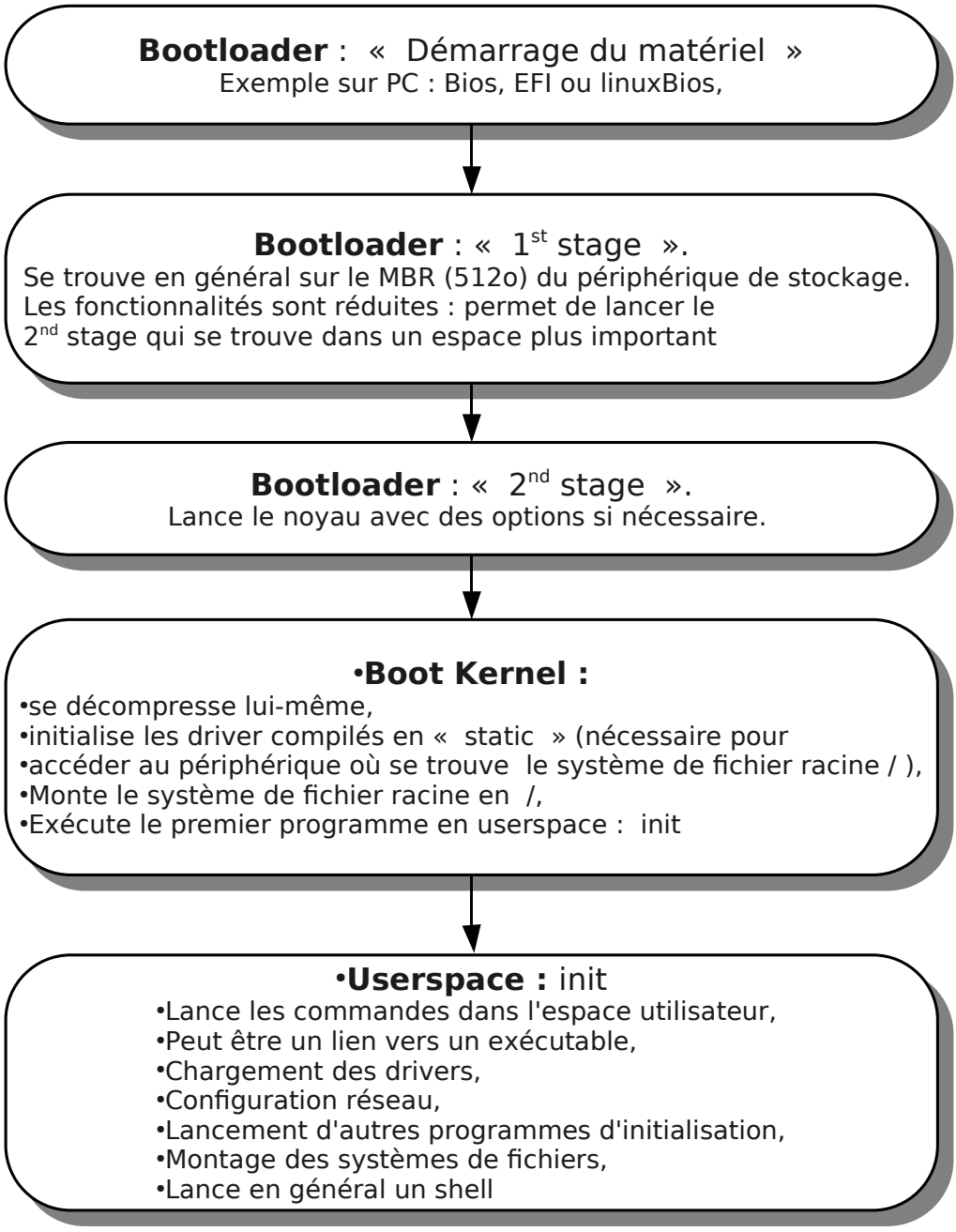
L'ordonnanceur (scheduler) : en multitâche, permet de gérer le temps imparti et l'ordre dans lequel sont exécutées chacune des tâches.

La MMU : permet de protéger des plages mémoires (translation d'adresses virtuelles en adresses physiques) afin qu'aucun programme n'accède à une zone qui ne lui est pas réservée. Dans le cas d'un dépassement, une interruption est envoyée et résulte par un crash de l'application ('segmentation fault').

7 *http://fr.wikipedia.org/wiki/Noyau_(informatique)*

Démarrage d'un système GNU/Linux.

1. De façon générale.

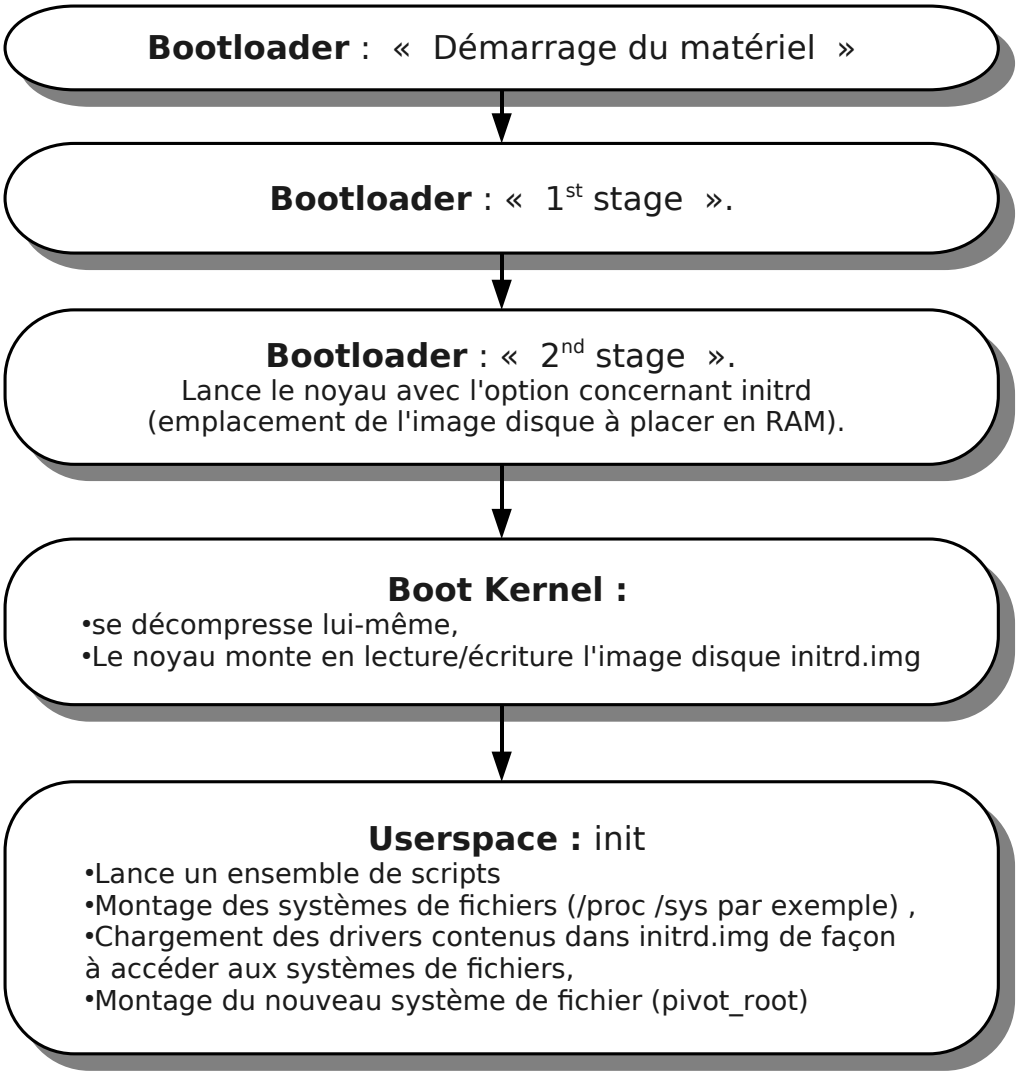


2. Cas de l'utilisation de initrd (Disque RAM initialisé au démarrage du système).[8]

Permet d'avoir les modules disponibles avant le montage du système racine et permet ainsi d'éviter de les compiler en dur (« static »). Cf drivers de systèmes de fichiers. Permet dans les distributions d'avoir un noyau standard et de ne modifier que le initrd.img pour s'adapter au choix de l'utilisateur lors de l'installation.

Cette option doit être précisée au moment de la compilation du noyau.

8 *Voir la manpage de initrd ainsi que initrd.txt dans la documentation du noyau.*



3. Init.

Lors du démarrage, init place le système dans un niveau d'exécution donné (c'est un à dire qu'un ensemble de scripts correspondants à ce niveau sont exécutés).

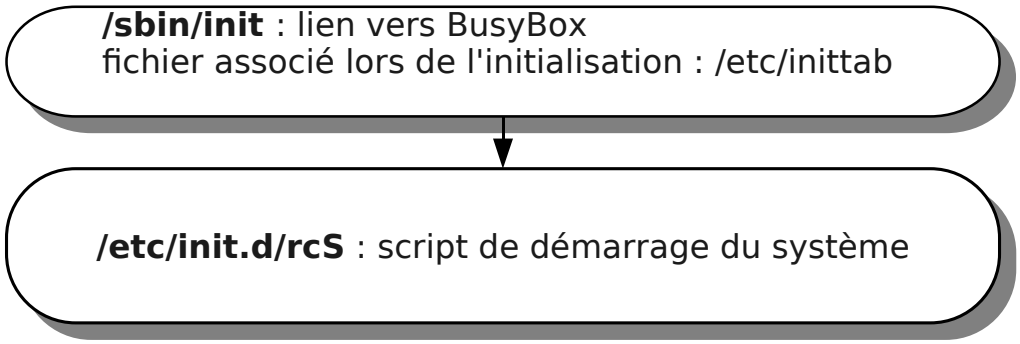
Descendant d'unix système V, les niveaux sont définis de la sorte :

- **0 :** Hors service : l'alimentation peut alors être coupée sans danger
- **1** : Mode mono-utilisateur (pour administration système)
- **2** : Mode multi-utilisateurs : fonctionnement normal sans NFS (identique au niveau 3 mais sans les fonctionnalités réseau).
- **3** : Mode multi-utilisateurs : fonctionnement normal pour systèmes en réseau, partageant leurs ressources avec d'autres systèmes.
- **4** : Inutilisé
- **5** : X11, interface graphique lancée
- **6** : Mise hors service et redémarrage : sert durant le redémarrage du système à partir d'un niveau de fonctionnement (2, 3, 4, 5). Le système passera ensuite au niveau 0.

Les scripts associés à ces niveaux sont définis dans le fichier /etc/inittab.

Dans notre cas, nous utiliserons généralement busysbox[9] comme application première lancée au démarrage. Cette dernière propose une version allégée des niveaux d'exécution (voir la documentation). L'initialisation du système se fait alors avec /etc/init.d/rcS.

9 *http://www.busybox.net/*



Ainsi , dans ce cas, on aura juste à créer le dernier script de façon à réaliser la configuration de la cible.

Exemple de script /etc/init.d/rcS. On désire monter les pseudo systèmes de fichiers /proc et /sys, ainsi que configurer l'interface réseau eth0 avec l'adresse 192.168.0.1. Proposer un script.

```
#!/bin/sh
.....
.....
.....
.....
.....
.....
.....
.....
```

Remarque : ifconfig <interface> <ip_address> [netmask <netmask>]

4. Les principaux bootLoader.

Pour x86 :

- ***LILO*** : Linux Loader (http://lilo.go.dyndns.org/)
- ***GRUB*** : Grand Unified Bootloader from GNU. (http://www.gnu.org/software/grub/). Nous privilégierons ce BootLoader : il offre un support réseau et permet des modifications sans être obligé de relancer la commande (particulièrement pratique lors des phases de test).
- ***Xosl***
- ***Chos***

Pour ARM :

- ***RedBoot :*** utilisé sur les cartes EP930x Olimex (provint du projet eCos). Permet l'utilisation du réseau
- ***Uboot*** : bootloader pour ARM

Noyau, périphériques & drivers.

1. Les périphériques.

La majorité des périphériques sont vus sous linux comme des fichiers se trouvant dans /dev.

On distingue les périphériques de type « character » des périphériques de type « block » :

- *périphériques de type « character »* : l'information est transmise par octets. L'accès au périphérique est de type séquentiel. Exemple : clavier, ports parallèles et série, son, video ...
- *périphériques de type « block »* : l'information est transmise par paquets d'octets, d'une taille donnée. L'accès au périphérique se fait sans ordre pré-établi. Exemple : disques durs, lecteur disquette, disque ram ...

Certains périphériques ne correspondent pas à des fichiers du répertoire /dev :

- les interfaces réseaux (eth0, eth1, ppp0...) : une interface réseau ne s'appuie pas forcément sur un périphérique physique. L'interface loopback lo en est un exemple.
- les drivers USB, firewire car ils s'appuient sur des drivers existants (ex : usb mass-storage qui utilisera en « émulation » des périphériques de type SCSI /dev/sdXX).

2. Drivers et périphériques.

Les fichiers du répertoire /dev sont définis par[10] :

- leur nom,
- leur type block ou character : b ou c,
- un nombre appelé « major » : associé à chaque driver,
- un nombre appelé « minor » : associé à chaque périphérique.

Exemple de périphériques de type « character » :

```
$ ls -l tty*
crw-rw-rw-  1  root tty   5,  0  jan 28 08:20 tty
crw-rw----  1  root root  4,  0  jan 28 08:21 tty0
crw-------  1  root root  4,  1  jan 28 08:22 tty1
crw-rw----  1  root tty   4, 10  jan 28 08:20 tty10
```

Exemple de périphériques de type « block » :

```
$ ls -l /dev/hda*
brw-rw----  1  root root  3, 0  jan 28 08:20 /dev/hda
brw-rw----  1  root root  3, 1  jan 28 08:20 /dev/hda1
brw-rw----  1  root root  3, 2  jan 28 08:20 /dev/hda2
brw-rw----  1  root root  3, 5  jan 28 08:20 /dev/hda5
brw-rw----  1  root root  3, 6  jan 28 08:20 /dev/hda6
```

Le répertoire /proc permet d'obtenir les informations liant les périphériques aux numéros de drivers utilisés :

```
#cat /proc/devices
Character devices:
  1 mem
```

10 *Voir la documentation du noyau : Documentation/devices.txt*

```
2 pty
3 ttyp
4 /dev/vc/0
4 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
...
```

Remarque : Certains numéros « major » sont liés de façon statique aux périphériques les plus usuels(voir Documentation/devices.txt du moyau). Dans d'autres cas, on peut utiliser une allocation dynamique d'un nombre « major ». Dans ce cas, un script de chargement du driver doit :

- déterminer un numéro major libre via /proc/devices,
- créer les fichiers périphériques correspondant,
- gérer les autorisations d'accès au périphérique.

Pour créer un fichier de périphérique, on utilise la commande :

```
mknod /dev/<device> [c|b] <major> <minor>
Exemple : mknod /dev/console c 5 1
```

1. Compilation sous forme de module.

La compilation sous forme de modules permet d'avoir des binaires séparés du noyau. Les avantages de cette méthode :

- on peut distribuer des drivers en version binaire seulement, (cependant, cela nécessite une nouvelle distribution en cas d'incompatibilité avec le noyau),
- permet d'être chargé en mémoire seulement lorsqu'ils sont nécessaires, puis déchargés,
- permet de réduire la taille du noyau,
- permet de tester les drivers durant les phases de développement sans avoir à relancer le noyau ...

Les modules sont installés dans le répertoire /lib/modules/<linux-version>/

La gestion des modules se fait à l'aide des commandes :

modprobe : chargement « intelligent », tenant compte des dépendances entre les modules (utilise le fichier /lib/modules/<linux-version>/modules.dep). On peut passer des paramètres à modprobe,

depmod : crée le fichier modules.dep qui définit les dépendances entre les différents modules (fichier modules.dep). Il suffit de lancer depmod -a après la compilation d'un nouveau module pour mettre à jour le fichier.

insmod : chargement d'un module en mémoire. On peut passer des paramètres à insmod.

rmmod : enlève un module de la mémoire.

modinfo : informations sur le module (licence, description, paramètre.

lsmod : liste l'ensemble des modules chargés en mémoire. Exemple :

```
# lsmod
Module                  Size  Used by
ext3                  123304  2
jbd                    48344  1 ext3
.....
```

1. informations fournies par le noyau.

4.1. dmesg ou cat /var/log/dmesg

Utile pour découvrir les actions du noyau depuis son démarrage (modules chargés en fonction des périphériques)

```
#dmesg
....
serio: i8042 AUX port at 0x60,0x64 irq 12
serio: i8042 KBD port at 0x60,0x64 irq 1
Serial: 8250/16550 driver $Revision: 1.90 $ 8 ports, IRQ sharing enabled
ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
.....
```

4.2.1./proc et /sys.

Par exemple :

- informations sur le CPU

```
#cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 2
....
```

- La liste des ports d'i/o : /proc/ioports

```
# cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
....
```

- ... etc ...

4.3.1. /var/log/

De plus, par l'intermédiaire de /var/log/syslog ou /var/log/messages on peut obtenir différentes informations sur le fonctionnement du noyau.

2. Compilation du noyau.

- Récupérer les sources sur le site http://www.kernel.org (éviter les noyaux fournis par les distributions car ils sont modifiés de façon conséquente),

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-<version>.tar.bz2
```

- Récupérer le fichier signature afin de vérifier l'intégrité du noyau téchargé,

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-<version>.tar.bz2.sign
```

- Récupérer une clé publique des développeurs de noyau[11]

```
gpg --keyserver pgp.mit.edu --recv-keys 0x517D0F0E
```

- vérifier l'intégrité du noyau téléchargé.

```
gpg --verify linux-<version>.tar.bz2.sign linux-<version>.tar.bz2
```

- décompacter le noyau :

```
tar jxvf linux-<version>.tar.bz2
```

- Si nécessaire, télécharger des *patch* pour le noyau.

La commande *patch* permet de mettre à jour les fichiers existants, de créer et supprimer des répertoires et fichiers.

```
cd linux-2.6.10
bzcat ../patch-2.6.11.bz2 | patch -p1
cd ..
mv linux-2.6.10 linux-2.6.11
```

Remarque 1 : utiliser *zcat* dans le cas d'un patch en .gz

Remarque 2 : Extrait d'un *patch* :

```
diff -Nru a/Documentation/DocBook/via-audio.tmpl b/Documentation/DocBook/via-audio.tmpl
```

-p1 signifie que l'on ne tient pas compte des a/ et b/ . Ainsi, avant d'appliquer le patch, il faut se placer dans le répertoire linux-<version>. -p<n> signifie que n est le nombre de niveaux de répertoires à ignorer pour l'application du patch

- Changer le *Makefile* afin de personnaliser le noyau par extraversion

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 14
EXTRAVERSION = .mon_noyau
```

- utiliser ccache afin d'optimiser la compilation (dans le *Makefile*).

```
CC              = ccache $(CROSS_COMPILER)gcc
HOSTCC          = ccache gcc
```

- Configurer le noyau.
  Le fichier de configuration du noyau se trouve dans le répertoire des sources et se nomme *.config*
  Le fichier de configuration des noyaux utilisés se trouve dans */boot* généralement
  Il est souvent pratique de partir d'une configuration vierge :

```
make allnoconfig.
```

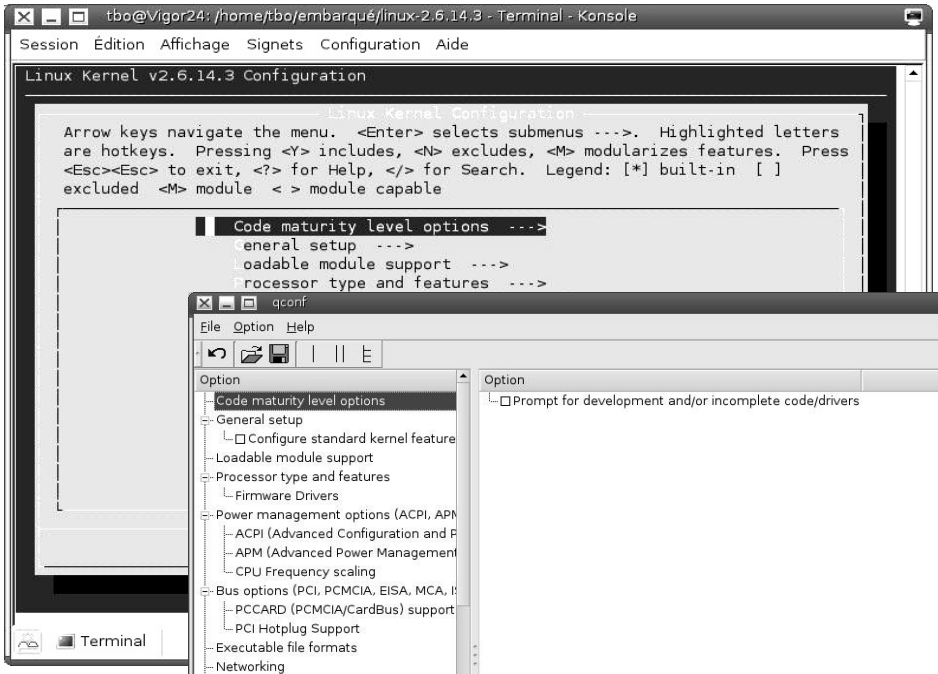
Pour lancer la configuration du noyau :

11 *http://www.kernel.org/signature.html*

make config : il faut répondre à toute les questions

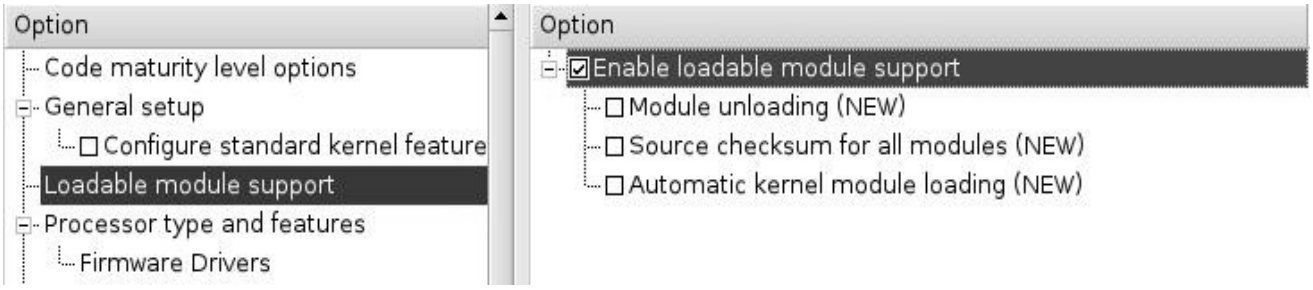
```
...
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [N/y/?]
....
```

make menuconfig : configuration avec interface ncurse

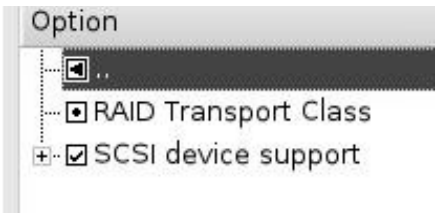


make xconfig : une configuration en mode graphique (le plus pratique)

On choisira entre autre, si on le souhaite le support des modules dans le noyau :



Par la suite, suivant le nombre de fois que l'on clique sur un driver, il peut apparaître comme un module.



RAID Transport class est choisi en tant que module.

SCSI device support est choisi en tant que driver compilé de façon statique.

Les commandes de configuration du noyau :	
make allnoconfig :	presque tout est mis à Non (feuille vierge)
make oldconfig :	permet de récupérer un ancien fichier .config.
make config :	configuration pas à pas,
make menuconfig :	configuration avec menu
make xconfig :	configuration avec interface graphique
make clean :	pour recompiler des drivers. Nettoie les fichiers générés.
make mrproper :	supprime tous les fichiers créés, y compris .config !!!

- compilation du noyau

```
make
ou
make -j <nbre_processeur>
```

Les fichiers générés :

- *vmlinux* : image du noyau non compressée,
- *arch/<arch>/boot/bzImage* : image du noyau compressée (big zipped). Image utilisée par défaut sur architecture x86.
- *arch/<arch>/boot/zImage* : image du noyau compressée (zlib). Image utilisée par défaut sur architecture arm.

- installation du noyau

```
make install
```

Installe les fichiers :

- */boot/vmlinuz-<version>* : image compressée du noyau,
- */boot/System.map-<version>* : tables des adresses des symboles du noyau,
- */boot/initrd-<version>.img* : Ram disque initiale (de boot) générée via mkinitrd.
- */etc/grub.conf* ou */etc/lilo.conf* : mise à jour des bootloader.

Cependant, dans le cas d'un développement pour un système embarqué, la plateforme de développement n'est pas forcément celle où on veut installer le noyau. Il existe 2 solutions :

- copier l'ensemble de ces fichiers sur la cible en respectant l'arborescence et les droits

```
cp -a -r  source destination (-a préserve les attributs)
```

- modifier le *Makefile* :

```
export INSTALL_PATH ?=/mnt/cible/boot
```

- installation des modules

```
make modules_install
```

Les modules sont installés dans */lib/modules/linux-<version>/* en respectant la même arborescence que le noyau. Les fichiers générés sont :

- *modules.aliases* : alias de modules pour le chargement des modules,
- *modules.dep* : dépendance entre les modules (peut être généré par *depmod -a*),
- *modules.symbol*

De même, si la cible de développement est différentes de la plateforme de développement, on peut modifier le lieu d'installation des modules via la ligne de commande :

```
make modules_install INSTALL_MOD=/mnt/cible/
```

Choix d'un système de fichier & périphériques de stockage.

1. Les différents systèmes de fichiers utilisés en embarqué.

***Préambule** : avant d'utiliser l'un des systèmes de fichiers, il est nécessaire de vérifier que :*

- *le module est compilé au niveau du noyau,*
- *le bootloader est capable de gérer ce type de système de fichier.*

1.1.1.Les systèmes de fichiers pour « block devices ».

Pour un système embarqué, on privilégie les systèmes de fichiers jounalisés :

- le système reste stable même après un crash ou un arrêt brutal,
- les données sont d'abord écrites dans le journal avant d'être inscrites sur le disque,
- inconvénient 1 : en cas d'arrêt brusque, le journal ne traite pas les transferts incomplets : il y a risque de perte des opérations qui étaient en cours.
- Inconvénient 2 : le journal occupe un espace non négligeable.

Différents systèmes de fichiers :

- ***Ext3*** : modification du système de fichier Ext2 (système de fichier historique) avec journalisation.
- ***reiserfs*** :système de fichier journalisé
- ***JFS***, ***XFS*** ....

On utilisera les systèmes de fichiers journalisés **ext3** ou **reiserfs** pour un stockage sur « *block device* » de type :

- Disque dur,
- Compact Flash, vues comme des périphériques IDE ou PCMCIA,
- ram disk,
- Disk On Module, DoM de la société PQI. Se comporte comme un disque IDE,
- Flash disk ide

1.2.1.Les systèmes de fichiers en lecture seule.

- ISO 9669 : pour les CDROM,
- CramFS : système de fichier prévus pour les système à base de ROM (taille maxi 256 Mo pour des fichiers de taille maximum 16 Mo). Le système utilise l'algorithme de compression zlib.

On utilisera le système de fichiers **CramFS** pour

- tout système mis en Rom
- pour une partition où l'on veut des données inaltérables.

1.3.1.Les systèmes de fichiers pour périphériques de type MTD (Memory Technology Device)[12]

Les mémoires de masse de type Flash comme les DoC (Disk On Chip) possède un Bios qui permet d'améliorer leur durée de vie par rapport au mémoires de type CompactFlash. Ils ne sont ni des « character devices », ni des « block devices ». On utilisera donc un pilote particulier, fourni par le projet MTD qui permettra de voir les mémoires comme des « block devices ».

Le système de fichier utilisé est le système JFFS2 (Jounalised Flash File System v2) :

12 *http://www.linux-mtd.infradead.org/*

- prévu pour utiliser les différents secteurs de façon homogène (les mémoire flash ont une limite du nombre de fois que sont inscrites les données)
- compressé afin d'utiliser au maximum les mémoires flash

Remarque : il faut éviter d'utiliser le pilote propriétaire car il ne peut être intégré au noyau : il obligera l'utilisation d'un ram disk au démarrage du système).

On utilisera le système de fichiers **JFFS2** pour mémoire flash. On compilera le noyau avec les modules du projet MTD de façon à pouvoir les utiliser comme des « block devices ». Ne jamais utiliser un système de fichiers de type Ext3 ou ReiserFS : le journal se trouvant dans un endroit fixe, il réduirait la durée de vie du composant.

2. Manipulation et création des systèmes de fichiers.

2.1.1.Vérifier l'intégrité d'un système de fichiers.

fsck.ext3 /dev/hda1 (fsck.<type>) pour vérifier la partition 1 (de type ext3) du disque hda, premier disque IDE. Le disque doit être démonté pour effectuer cette opération

2.1.2.Créer un système de fichiers.

mkfs.ext3 /dev/hda1 (mkfs.<type>)

2.1.3.(Dé)monter un système de fichiers.

Montage d'une clé USB.

mkdir /mnt/usbkey	création du point de montage,
mount -t auto /dev/sda1 /mnt/usbkey	montage de la clé
umount /mnt/usbkey	démontage de la clé

On peut monter aussi une image d'une partition (*loopback device*)

cp /dev/sda1 cle_usb.img	Création de l'image disque d'une partition en vfat
mount -o loop -t vfat cle_usb.img /mnt/usbkey	Montage du système de fichier.

2.1.4.Utiliser des images disques lors du développement.

Créer une image vide de 64 Mo :

```
dd if=/dev/zero of=disk.img bs=1024 count=65536
```

Formater cette image en ext3 :

```
mkfs.ext3 disk.img
```

Monter cette image :

```
mount -o loop -t ext3 disk.img /mnt/disk
```