Embarquez Linux!

ou Linux Everywhere...

Pierre Ficheux (pierre.ficheux@openwide.fr)

avec la participation de

Patrice Kadionik (kadionik@ensierb.fr)

Septembre 2004

Résumé

Le but de cet article est de décrire quelques techniques utilisables pour embarquer une distribution Linux réduite sur divers supports de faible capacité (CompactFlash, Disk On Module ou *DoM*, clé USB, Disk On Chip ou *DoC*). Après avoir présenté les différents supports étudiés, nous détaillerons une structure de système permettant de garantir une sécurité maximale de l'installation tout en minimisant l'empreinte mémoire.

Cette article fait suite aux diverses publications de l'auteur ainsi qu'à l'influence de collègues, amis et complices de longue date qui se reconnaîtront facilement :-) Pour l'heure je remercie Patrice Kadionik pour sa contribution concernant la technologie des mémoires et Michel Stempin pour ses commentaires

Les exemples présentés se basent sur un noyau Linux 2.4 - encore majoritairement répandu - mais peuvent être très simplement adaptés à un noyau 2.6.

La technologie des mémoires pour l'embarqué

Linux est indissociable de la mémoire qu'il utilise. Dans un système embarqué, la mémoire est une denrée rare qu'il faut économiser. Il convient maintenant de regarder les différentes technologies de mémoire que l'on peut utiliser sous Linux pour l'embarqué.

On distingue généralement deux familles de mémoire :

- La mémoire RAM (*Random Access Memory*).
- La mémoire ROM (*Read Only Memory*).
- Les paramètres essentiels pour caractériser une mémoire sont :
- Sa capacité c'est à dire la quantité d'information qu'elle peut stocker (en octets).
- Son temps d'accès c'est à dire le temps qu'il faut pour obtenir une donnée (en secondes). Pour les mémoires synchrones, on définit la fréquence d'accès comme l'inverse du temps d'accès (en Hertz). La bande passante de la mémoire est alors le produit fréquence-taille des données (en octets) exprimée en octets/s.
- Le type de boîtier, le coût, la durée de vie, etc.

Les mémoires RAM

Les mémoires RAM sont des mémoires pour lesquelles les opérations de lecture et d'écriture sont autorisées. Appelées mémoires vives, elles sont volatiles c'est à dire que l'on peut perdre l'information qu'elles contiennent (données) en cas de coupure ou microcoupure d'alimentation. Le temps d'accès aux données est faible (quelques ns à quelques dizaines de ns en général). Elles sont généralement utilisées comme mémoire centrale du système embarqué. On distingue alors

différents types de mémoires RAM :

- La mémoire statique SRAM (*Static RAM*). L'information (un bit) est mémorisée dans un dispositif bistable à base de transistors (en technologie MOS généralement). L'information est conservée tant que la mémoire est alimentée. La mémoire SRAM est rapide, chère et de faible capacité (quelques Mo). Le temps d'accès est de quelques ns.
- La mémoire dynamique DRAM (*Dynamic RAM*). L'information (un bit) est mémorisée dans la capacité d'entrée d'un transistor MOS (Cgs) et donc est volatile, ce qui oblige à recharger périodiquement cette capacité sous peine de perdre l'information : c'est le rafraîchissement de la mémoire. La mémoire DRAM est moyennement rapide, très bon marché et de forte capacité (quelques dizaines de Mo). Le temps d'accès est de quelques dizaines de ns. Pour diminuer le temps d'accès, on a développé la mémoire SDRAM synchrone (*Synchronous DRAM*). Le transfert de données se fait au rythme d'un signal d'horloge. Le temps d'accès est de quelques ns dans le meilleur des cas. On retrouve ici les différentes types de mémoires utilisées dans un PC d'hier et d'aujourd'hui : mémoire EDO, SDRAM, DDR SDRAM (*Double Data Rate SDRAM*)...

Dans un système embarqué, la mémoire RAM sera utilisée comme mémoire centrale. Elle peut être aussi structurée pour former un système de fichiers Linux, on parlera alors de *RAMdisk*. Cela peut être intéressant pour stocker des fichiers contenant des données d'acquisition qui pourront être ensuite transférés par le réseau, des fichiers temporaires comme les fichiers de log.

Les mémoires ROM

Les mémoires ROM sont des mémoires où seules les opérations de lecture les opérations de lecture sont autorisées. Avec les différentes évolutions technologiques, on distingue différents types de mémoires ROM :

- La mémoire ROM. Elle est programmée une seule fois définitivement lors de sa fabrication en usine et est réservée aux productions de grande série et donc inaccessible à l'utilisateur.
- La mémoire PROM (*Programmable ROM*). Elle peut être programmée une seule fois par l'utilisateur cette fois-ci à l'aide d'un programmateur de PROM. Elle est aussi appelée PROM à fusibles et n'est pratiquement plus employée.
- La mémoire EPROM (*Erasable PROM*). Cette fois-ci, la mémoire EPROM peut être reprogrammée à condition de l'effacer au préalable en la laissant bronzer sous une lampe à UV une vingtaine de minutes. On l'appelle aussi UV-PROM ! Elle est programmée avec un programmateur nécessitant une tension de programmtion élévée de 12,5 à 21 V en respectant un algorithme de programmation. La durée de rétention de l'information est de 10 ans au moins si l'on a occulté la petite fenêtre transparente pour les UV sur le boîtier avec une étiquette opaque. La mémoire EPROM supporte quelques milliers de cycles de reprogrammation. Elle est lente (quelques centaines de ns à quelques dizaines de ns), assez chère et de capacité moyenne (quelques centaines de Ko).
- La mémoire EEPROM (*Electricaly Erasable PROM*). C'est une amélioration technologique de la mémoire EPROM. La mémoire EEPROM peut être effacée et reprogrammée électriquement (pas besoin d'UV) avec une tension de programmation faible (5 V, c'est à dire l'alimentation du système) en respectant la aussi un algorithme de programmation qui peut être implémenté de manière logicielle. C'est donc un point très intéressant car la reprogrammation peut se faire in-situ en cours de fonctionnement du système. Le temps de programmation (généralement un octet) est important (supérieur à 10 ms), ce qui l'un des points faibles de la mémoire EEPROM. La durée de rétention de l'information est là aussi de 10 ans au moins. Comme précédemment, la mémoire EEPROM supporte quelques milliers de cycles de reprogrammation. Elle est lente (quelques centaines de ns à quelques dizaines de ns), assez chère et de capacité moyenne. Elle tend à être remplacée par la mémoire FLASH.

- La mémoire FLASH. C'est en fait à la base une mémoire de type EEPROM et donc en possède les principales caractéristiques mais avec des améliorations technologiques notables. Le temps de reprogrammation (d'un octet) est d'une centaine de µs ou moins (d'où son nom). Elle supporte plusieurs centaines de milliers de cycles de reprogrammation. Elle est bon marché et de très forte capacité (quelques centaines de Mo voire plus) surtout depuis l'explosion du marché des appareils photos numériques. On la retrouve mise en oeuvre dans les cartes CompactFlash, Memory Stick, etc.Il faut enfin noter qu'il existe des mémoires FLASH de type NOR et de type NAND. La mémoire FLASH NOR est la plus utilisée car plus simple à mettre en oeuvre et plus rapide.

Dans un système embarqué, la mémoire FLASH (sous forme de boîtiers électroniques, de cartes CompactFlash, de boîtiers DoC) est de loin la plus utilisée pour sa forte capacité, son faible coût, son faible temps d'accès et sa facilité de reprogrammation. On l'utilise pour y stocker le programme de démarrage (ou *bootloader*), le noyau Linux mais aussi pour y mettre des systèmes de fichiers Linux accessibles en lecture seulement mais aussi en écriture. Dans ce dernier cas, on pourra utiliser des systèmes de fichiers journalisés plus robustes aux coupures d'alimentation comme EXT3 mais aussi JFFS2 (*Journalling Flash File System version 2*) prévu pour mieux « résister » aux coupures d'alimentation durant la phase de reprogrammation de la mémoire FLASH. Enfin, il est aussi possible d'exécuter directement le noyau Linux depuis la mémoire FLASH sans le recopier puis le décompresser en mémoire RAM en utilisant la technique XIP (*eXecute In Place*).

Créer une distribution Linux embarqué

Le sujet est traité de manière approfondie dans les publications citées. En fait diverses solutions sont possibles.

1. Réduire une distribution classique (type Red Hat ou Mandrake)
2. Utiliser une distribution réduite existante, commerciale ou non (type LRP - Linux Router Project - ou MontaVista Linux)
3. Construire la distribution à partir des composants (approche LFS, Linux From Scratch)

La première solution est complexe et ne donne en général pas de résultats satisfaisants car les distributions classiques ne sont pas conçues pour être réduite dans des proportions acceptables pour le sujet qui nous intéresse (soit une empreinte mémoire inférieure à 10 Mo). La deuxième est relativement simple à mettre en oeuvre et sera adaptée dans le cas de l'utilisation ponctuelle de Linux pour une application embarquée.

Dans les publications précédentes, nous avions choisi d'utiliser la troisième solution, soit de construire une distribution à partir de composants fondamentaux (noyau Linux, GLIBC, etc.). Cette approche nécessite un investissement intellectuel non négligeable mais elle a l'avantage d'être très pédagogique et d'assurer une parfaite maîtrise du résultat. De plus, une connaissance correcte de la structure du système Linux permet de mettre en place assez facilement un système de démonstration fournissant un noyau adapté et un environnement minimal fonctionnel (système minimal démarrant un interpréteur de commande type `bash`).

Cependant, la création de la distribution n'est pas le sujet qui nous intéresse aujourd'hui et nous nous focaliserons de ce fait sur l'utilisation de périphériques exotiques ou du moins quelque peu spécifiques en tant que mémoire de masse.

Mémoire de masse et système de fichier principal

Dans un système Linux - réduit ou non - la mémoire de masse a deux fonctionnalités principales:

1. Héberger le système de fichier principal (ou *root filesystem*) de Linux. L'existence de ce système de fichier est indispensable au bon fonctionnement du système.
2. Le plus souvent permettre le démarrage du système en accueillant un programme spécialisé (comme LILO ou GRUB) dans le premier secteur du périphérique concerné.

Dans la majorité des cas, la mémoire de masse est constituée d'un disque dur, accessible à travers les fichiers spéciaux (ou *block devices*) `/dev/hdX` pour les disques IDE ou `/dev/sdX` pour les disques SCSI. La variable *X* pourra prendre les valeur a, b, c, d ou plus suivant la position du disque sur le bus concerné. Nous verrons dans la suite de l'article que certains périphériques permettront d'utiliser les mêmes fichiers de manière transparente car le noyau Linux et/ou le matériel présenteront à l'utilisateur une interface compatible même si physiquement nous n'avons pas affaire à de véritables disques IDE ou SCSI. En revanche, certains autres périphériques comme les DoC utiliseront des pilotes différents mais qui conserveront leurs caractéristiques de périphériques en mode bloc et seront donc utilisables de la même manière pour les opérations habituelles sur les mémoires de masse.

- Le *partitionnement* avec l'outil `fdisk`
- Le *formatage* avec l'outil `mke2fs` (en cas d'utilisation d'un système de fichier de type EXT3)
- Le *montage* avec `mount`

Concernant le type de système de fichier, le choix d'un système de fichier journalisé type EXT3 est fortement conseillé sachant qu'un système embarqué est le plus souvent arrêté sur une coupure d'alimentation. Concernant l'intégrité des données, il y a deux niveaux à considérer :

- l'intégrité d'écriture d'un bloc : en cas de coupure, un bloc peut-il être partiellement écrit ou non ?
- l'intégrité d'écriture d'un fichier : en cas de coupure, un fichier peut-il être partiellement écrit ou non ?

Il existe des mémoires CompactFlash qui ne garantissent pas le premier point mais par contre ceci est garanti dans le cas des DoC. Le deuxième point est intégralement dépendant du type de système de fichier.

Cependant, le choix de tel ou tel système ne garantira pas la sécurité des données à 100%. Si le système nécessite une garantie absolue de fiabilité, il sera nécessaire de mettre en place une procédure matérielle de traitement de la coupure. Dans la suite de l'article, nous donnerons cependant un méthode permettant de minimiser les risques de perte de données en utilisant un système de fichier principal monté en lecture seule.

Les mémoires CompactFlash

La CompactFlash est une mémoire FLASH permanente de type NAND. Elle a été popularisée par le développement des appareils photographiques numériques même si aujourd'hui d'autre supports plus récents sont proposés en remplacement (SD, Memory Stick, etc.). Du fait de son développement au niveau du grand public, le coût de la CompactFlash est relativement faible (environ 15 euros pour une mémoire 16Mo) et elle est disponible pour de grandes capacités (1 Gb). Par contre il ne faut pas oublier que ce type de mémoire subit une usure non négligeable en cas d'écritures répétées. Il conviendra donc de définir correctement l'architecture du système afin de limiter cette usure.

Au niveau des systèmes informatiques embarqués, elle est le plus souvent utilisée de plusieurs manières:

- Comme un disque IDE (cas le plus courant) soit directement soit à travers un adaptateur PC-Card
- Via un adaptateur USB

Les systèmes dédiés type mini-PC (VIA C3 ou autres compatibles x86) proposent souvent un connecteur CompactFlash directement sur la carte mère et permettant l'utilisation via une interface IDE. Dans ce cas l'accès est totalement identique à un disque dur IDE et il n'est pas nécessaire d'ajouter un quelconque pilote au noyau Linux (à part valider le support IDE). De même, on pourra installer les programmes de démarrage classiques comme LILO de la même manière.

Dans le cas d'un PC classique, on peut ajouter facilement un adaptateur que l'on pourra connecter par un câble IDE comme décrit sur la figure 1 ci-dessous.

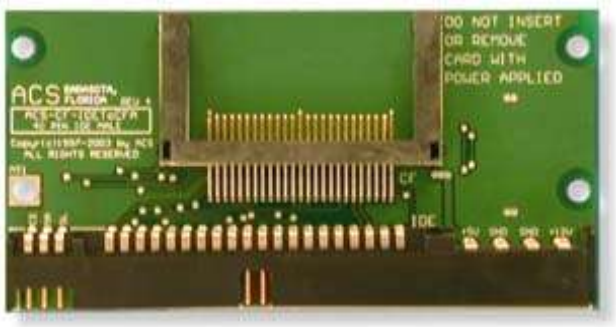

Figure 1: Adaptateur IDE pour CompactFlash

Pour un PC portable, on peut utiliser un adaptateur PC-Card qui ne nécessite par non plus de pilote spécifique mis à part la validation du support adéquat dans le noyau Linux (pilote PCMCIA). La figure 2 décrit un adaptateur de ce type, la figure 3 indique la configuration à ajouter au menu de configuration ATA/IDE/MFM/RLL du noyau Linux 2.4 afin de pouvoir utiliser ce type d'adaptateur. Grâce à un tel adaptateur (coûtant aux alentours de 15 euros), le PC portable peut devenir un station de développement permettant de configurer la CompactFlash avant installation sur la cible.

Figure 2: Adaptateur PC-Card pour CompactFlash

Figure 3: Validation sur support IDE CS

Lors de l'insertion de l'adaptateur équipé de la CompactFlash, on obtient la trace suivante dans le fichier `/var/log/messages`.

```
Sep 22 11:12:51 localhost cardmgr[830]: executing: 'modprobe ide-cs'
Sep 22 11:12:51 localhost kernel: ide2 at 0x100-0x107,0x10e on irq 3
Sep 22 11:12:51 localhost kernel: hde: attached ide-disk driver.
Sep 22 11:12:51 localhost kernel: ide_cs: hde: Vcc = 3.3, Vpp = 0.0
Sep 22 11:12:51 localhost cardmgr[830]: executing: './ide start hde'
```

Partionnement de la CompactFlash

La CompactFlash est donc visible à travers le device `/dev/hde` et l'on peut donc la partitionner, formater et monter comme suit. Pour le partitionnement, il est probable que la FLASH contienne déjà un système de fichier de type DOS, utilisé par les équipements grand public. Il conviendra donc de supprimer la partition DOS avant de créer la partition Linux.

```
# fdisk /dev/hde

Commande (m pour aide) : p

Disque /dev/hde : 4 têtes, 32 secteurs, 490 cylindres
Unités = cylindres sur 128 * 512 octets

Périphérique Amorce    Début       Fin    Blocs   Id  Système

Commande (m pour aide) : n
Action de commande
   e   Etendue
   p   Partition primaire (1-4)
p
Nombre de partitions (1-4): 1
Premier cylindre (1-490, 1 par défaut) :
Utilisation de la valeur par défaut 1
Dernier cylindre ou +size ou +sizeM ou +sizeK (1-490, 490 par défaut) :
Utilisation de la valeur par défaut 490

Commande (m pour aide) : p

Disque /dev/hde : 4 têtes, 32 secteurs, 490 cylindres
Unités = cylindres sur 128 * 512 octets

Périphérique Amorce    Début       Fin    Blocs   Id  Système
/dev/hde1             1       490     31344   83  Linux

Commande (m pour aide) : w
La table de partition a été modifiée !

Appel de ioctl() pour relire la table de partition.

AVERTISSEMENT: Si vous avez créé ou modifié
une partition DOS 6.x, reportez-vous au manuel de fdisk
pour plus d'informations.
Synchronisation des disques.
```

Formatage de la partition

Une fois partitionnée, on peut formater la partition en EXT3 en utilisant `mke2fs`.

```
# mke2fs -j /dev/hde1
mke2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
7840 inodes, 31344 blocks
1567 blocks (5.00%) reserved for the super user
First data block=1
4 block groups
8192 blocks per group, 8192 fragments per group
1960 inodes per group
Superblock backups stored on blocks:
        8193, 24577

Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Montage de la partition

On peut alors monter la partition et vérifier la taille disponible.

```
# mount -t ext3 /dev/hde1 /mnt/tmp
# df /mnt/tmp/
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hde1                30349      1043     27739   4% /mnt/tmp
```

On remarque que 1Mo sont déjà occupés, cela correspond au journal du système de fichier EXT3. La valeur de 1Mo est la plus petite taille envisageable pour EXT3 (la taille normale étant de 8 Mo). La taille du journal est donc calculée en fonction de la taille de la partition. Dans le cas de cette CompactFlash 32Mo, la taille est donc limitée à 1Mo. Pour connaître la taille exacte on peut utiliser l'outil `debugfs`.

```
# debugfs -R 'stat <8>' /dev/hde1
debugfs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Inode: 8   Type: regular    Mode:  0600   Flags: 0x0   Generation: 0
User:     0   Group:     0   Size: 1048576
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 2060
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x415293ec -- Thu Sep 23 11:14:20 2004
atime: 0x00000000 -- Thu Jan  1 01:00:00 1970
mtime: 0x415293ec -- Thu Sep 23 11:14:20 2004
BLOCKS:
(0-11):263-274, (IND):275, (12-267):276-531, (DIND):532, (IND):533, (268-523):53
4-789, (IND):790, (524-779):791-1046, (IND):1047, (780-1024):1048-1292
TOTAL: 1030
```

La taille est indiquée dans le champs *Size*, soit 1048576 octets.

Installation de LILO

L'installation de LILO est simple car il s'agit d'un périphérique IDE. Le principe est d'utiliser l'option `-r` de la commande `lilo` qui effectue un `chroot` sur la partition contenant la distribution embarquée (ce qui revient en gros à exécuter `lilo` comme si le système avait démarré sur cette partition). Le fichier `lilo.conf` a l'allure suivante:

```
prompt
timeout=50
boot=/dev/hde
map=/boot/map
install=/boot/boot.b
#disk=/dev/hde
#bios=0x80
image=/boot/bzImage-2.4.20
        label=Linux
        read-only
        root=/dev/hde1
```

Les lignes `disk` et `bios` devront être utilisées si `lilo` indique que la CompactFlash n'est pas vu comme premier disque par le BIOS (le code 0x80 identifie le premier disque géré par le BIOS). En validant ces deux lignes on indique que la CompactFlash est le premier disque vu du BIOS.

Les mémoires Disk On Module (DoM)

Le DoM est une mémoire FLASH associée à une interface IDE. Au niveau de l'encombrement, la taille n'est que légèrement supérieure à un connecteur IDE et le DoM peut donc être installé dans n'importe quel système équipé de connecteur IDE au format 3 pouces et demi. La figure ci-dessus montre un DoM 16Mo.

Figure 3: Disk On Module

Le module est bien entendu vu comme un disque IDE et la configuration est totalement similaire à celle décrite dans le paragraphe précédent pour la CompactFlash IDE.

Les clés USB

La clé USB est devenu un support très populaire de part sa très grande diffusion dans le monde de la bureautique, la vieille disquette 1.44 Mo étant devenue bien maigrichonne pour accueillir la plupart des fichiers actuels. Tous les utilisateurs de matériel informatique ont maintenant un clé USB dans la poche et on trouve même des couteaux suisses avec clé USB comme décrit sur la figure 4!

Figure 4: Couteau Suisse avec clé USB

Utilisation sous Linux

Il est bien entendu possible de l'utiliser sous Linux et elle sera alors vue comme un disque SCSI. La plupart des distributions actuelles intègrent le support des clés USB en natif (dans le noyau livré) et l'insertion de la clé provoque l'affichage du message suivant dans les traces du système.

```
Sep 21 12:13:26 localhost kernel: hub.c: new USB device 00:07.2-1, assigned
address 2
Sep 21 12:13:26 localhost kernel: usb.c: USB device 2 (vend/prod 0xea0/0x6803)
is not claimed by any active driver.
Sep 21 12:13:27 localhost kernel: SCSI subsystem driver Revision: 1.00
Sep 21 12:13:27 localhost kernel: Initializing USB Mass Storage driver...
Sep 21 12:13:27 localhost kernel: usb.c: registered new driver usb-storage
Sep 21 12:13:27 localhost kernel: scsi0 : SCSI emulation for USB Mass Storage
devices
Sep 21 12:13:27 localhost kernel:   Vendor: OTi       Model: Flash Disk
Rev: 1.11
Sep 21 12:13:27 localhost kernel:   Type:   Direct-Access
ANSI SCSI revision: 02
Sep 21 12:13:27 localhost kernel: USB Mass Storage support registered.
```

La clé est utilisable à travers le fichier spécial `/dev/sda` et l'on peut bien entendu la partitionner, formater, monter, etc. La clé suivante contient deux partitions de 16 Mo, l'une au format VFAT pour les transferts Windows, l'autre au format EXT3 contenant une mini-distribution Linux. L'existence de cette deuxième partition Linux ne trouble en rien l'utilisation de la clé sous Windows.

Démarrage sur la clé

Le démarrage sur la clé pose quelques petits problèmes:

1. Le BIOS du PC doit permettre le démarrage sur support disque USB. C'est le cas de la majorité des PC spécialisés pour les applications embarquées (EDEN, LEX, etc.) mais ce n'est pas le cas des PC grand public.
2. La détection de la clé USB lors du démarrage du noyau Linux est asynchrone par rapport au montage du système de fichier principal (ou *root filesystem*). De ce fait, un noyau Linux standard ne pourra pas utiliser comme *root filesystem* une partition d'une clé USB. Cependant la modification à apporter au noyau pour permettre cette utilisation est minime et se limite à un *patch* de quelques lignes appliqué au fichier `init/do_mounts.c`. Il existe plusieurs patch disponibles, celui-ci est accessible depuis le FAQ du site http://www.Linux-usb.org.

```
--- do_mounts.c   2003-09-03 20:27:20.000000000 +1000
+++ ../../../kernel-source-2.4.22/init/do_mounts.c   2003-12-15
08:58:40.000000000 +1100
@@ -348,6 +348,7 @@ static void __init mount_block_root(char
 {
      char *fs_names = __getname();
      char *p;
+     int retries = 0;

      get_fs_names(fs_names);
 retry:
@@ -369,8 +370,16 @@ retry:
            printk ("VFS: Cannot open root device \"%s\" or %s\n",
                  root_device_name, kdevname (ROOT_DEV));
            printk ("Please append a correct \"root=\" boot option\n");
-           panic("VFS: Unable to mount root fs on %s",
-                 kdevname(ROOT_DEV));
+           if (++retries > 5)
+                 panic("VFS: Unable to mount root fs on %s",
+                       kdevname(ROOT_DEV));
+           else {
+                 /* wait 1 second and try again */
+                 printk ("Retrying in 1 second, try #%d\n", retries);
+                 current->state = TASK_INTERRUPTIBLE;
+                 schedule_timeout(HZ);
+                 goto retry;
+           }
      }
      panic("VFS: Unable to mount root fs on %s", kdevname(ROOT_DEV));
 out:
```

Le principe de la modification est simple: on ajoute une attente de quelques secondes (maximum 5 essais) pour laisser le temps à la clé USB d'être détectée. Ce n'est pas des plus élégants mais c'est simple et cela fonctionne. Une modification similaire à ajouter à la fin de la fonction `mount_root()` est également disponible ci-dessous.

```
/*
 * Patch for USB boot
 */
          {
            /* begin jordi *************************** */
            static DECLARE_WAIT_QUEUE_HEAD (jordi_queue);
            printk ("\n\n\n--------------------------------\n");
            printk (" WAITING FOR A WHILE (1000) \n");
            printk (" TO DETECT THE USB DISK \n");
            sleep_on_timeout (&jordi_queue, 1000);
            printk ("--------------------------------\n\n\n");
            /* end jordi *************************** */
          }
/* End of patch */
        mount_block_root("/dev/root", root_mountflags);
}
```

Au niveau de la configuration du noyau, il faudra valider les différentes options (SCSI et USB) *en statique* pour permettre la détection de la clé USB en tant que disque au démarrage. Les figures suivantes indiquent la configuration SCSI et USB. Bien entendu si vous disposez d'un contrôleur OHCI et non UHCI, il faudra adapter la configuration USB.

Figure 5: Configuration SCSI du noyau

Figure 6: Configuration USB du noyau

Concernant l'installation de LILO, le principe est le même que pour les CompactFlash ou DoM (puisque c'est également un périphérique en mode bloc) sauf qu'il faut utiliser `/dev/sda` au lieu de `/dev/hda`.

Les mémoires Disk On Chip (DoC)

Le DoC est une mémoire FLASH intelligente (contenant un BIOS) développée par la société M-

Systems (http://www.m-sys.com). Le figure 7 présente l'aspect du DoC.

Figure 7: Le Disk On Chip de M-Systems

Du fait de l'existence de ce BIOS, les DoC ont une durée de vie plus élevée que les CompactFlash ou les DoM, mais ils sont également plus onéreux. De même, ils sont plus difficile à trouver et sont de ce fait réservés à des applications professionnelles.

Utilisation du DoC sous Linux

Les DoC utilisent une interface propriétaire (ni IDE, ni SCSI) et leur utilisation nécessite une configuration spéciale du noyau Linux. Il y a deux méthodes possibles:

1. Utiliser le pilote propriétaire fourni par M-Systems (un *patch* du noyau)
2. Utiliser le pilote MTD du noyau Linux

Il faut noter que l'utilisation du pilote propriétaire interdit en toute rigueur la construction d'un noyau incluant le pilote en statique car ce pilote contient des parties non GPL fournies uniquement en binaire (il faut alors passer par un *initrd*). De ce fait, dans la suite du document nous utiliserons uniquement le pilote MTD. L'utilisation de ce pilote est décrite en détails au chapitre 7 de l'ouvrage *Linux Embarqué* mais nous allons rappeler brièvement la procédure.

La validation du support MTD (pour *Memory Technology Devices*) s'effectue dans le menu correspondant de la configuration du noyau Linux comme décrit sur les figures 8 et 9.

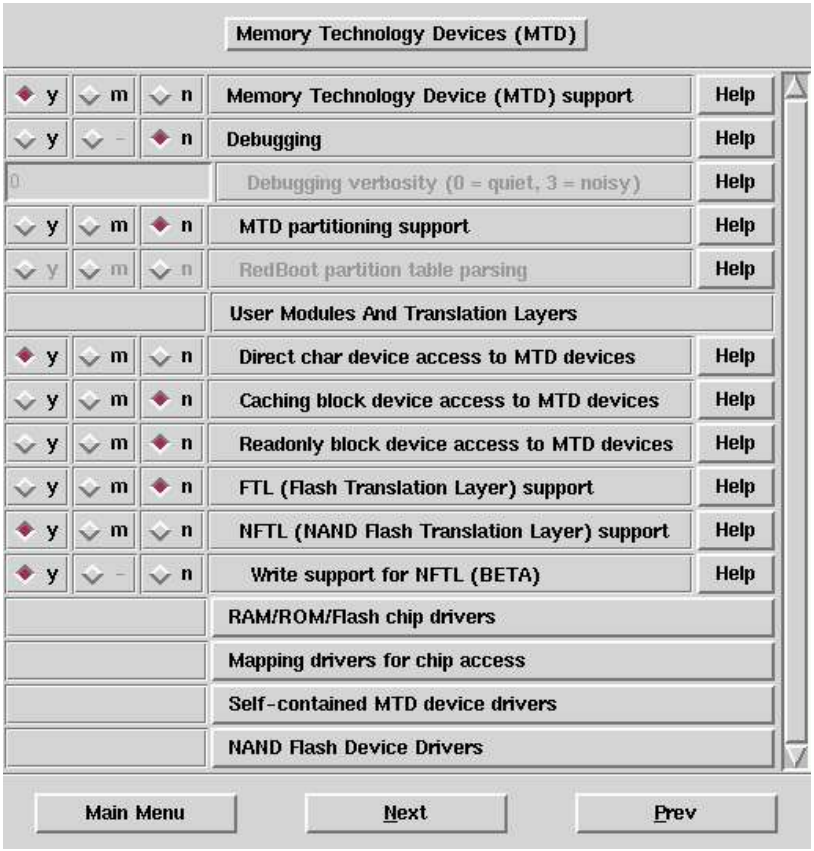

Figure 8: Validation du support MTD

Figure 9: Validation du support DoC dans MTD

Le DoC est également accessible à travers des fichiers spéciaux en mode bloc mais il est nécessaire de les ajouter au système. Pour ce faire, il faut récupérer l'archive du projet MTD sur le site http://www.Linux-mtd.infradead.org. via le serveur CVS. Dans le répertoire `mtd/util` ainsi créé on trouve le script `MAKEDEV` permettant la création des points d'entrée sous `/dev`. Le nommage des fichiers est similaire à l'IDE ou au SCSI soit `/dev/nftla` pour le premier périphérique et `/dev/nftla1` pour la première partition de ce périphérique. On peut bien entendu utiliser les programmes classiques de partitionnement, formatage et montage sur ces fichiers spéciaux.
Suite à l'installation du noyau contenant le support MTD et au redémarrage du système la détection du DoC doit être visible dans les trace. On peut également vérifier sa présence grâce à la structure `/proc`.

```
# cat /proc/mtd
dev:  size        erasesize name
mtd0: 02000000    00004000  "DiskOnChip 2000"
```

Par défaut, les DoC sont également livrés avec un système de fichier DOS, il est donc nécessaire de les formater pour l'utilisation en EXT3 ou autre système de fichier Linux. La procédure est identique sauf que l'on utilise le fichier spécial `/dev/nftla`.

Démarrage à partir du DoC

Le DoC a une géométrie spéciale qui n'a rien à voir avec un disque IDE ou SCSI. De ce fait il faut utiliser une version modifiée de LILO pour pouvoir démarrer sur ce support. Cette version adaptée (soit `lilo-mtd`) est disponible dans le répertoire `patches` des sources MTD chargées à partir du site. Le fichier `lilo.conf` adapté est très similaire aux fichiers précédents.

```
boot=/dev/nftla
install=/boot/boot.b-mtd
map=/boot/map
read-only
vga = normal
# End LILO global section
image = /boot/bzImage-2.4.20_mtd
        root = /dev/nftla1
        label = Linux
```

Si la partition du Doc est montée sur `/mnt/doc`, on installe le secteur de démarrage par une syntaxe identique à celle `lilo`.

```
# lilo-mtd -r /mnt/doc -v
Patched LILO for M-Systems' DiskOnChip 2000
LILO version 21, Copyright 1992-1998 Werner Almesberger

Reading boot sector from /dev/nftla
Merging with /boot/boot.b-mtd
Boot image: /boot/bzImage-2.4.20_mtd
Added Linux *
Backup copy of boot sector in /boot/boot.5D00
Writing boot sector.
```

Utilisation de CRAMFS

Après de ce tour d'horizon de quelque supports mémoire FLASH utilisables sous Linux, nous allons détailler la mise en place d'un système embarqué sécurisé basé sur l'utilisation du système de fichier en lecture seule CRAMFS. Je précise que la suggestion de l'utilisation de CRAMFS vient de mon collègue et ami Michel Stempin, grand spécialiste de Linux embarqué et auteur – entre autres - d'une publication sur le sujet dans Linux Magazine :-) (voir bibliographie)

Dans les paragraphes précédents, nous avons envisagé la construction de notre système dans une architecture classique, soit le système de fichier principal monté en lecture/écriture et utilisant un format de système de fichier journalisé (EXT3, JFFS2 ou autre). Cette structure est simple à mettre en oeuvre mais le système n'est pas à l'abri d'un incident vu qu'il sera la plupart du temps arrêté sur une coupure d'alimentation. De ce fait le système de fichier peut être endommagé et donc le redémarrage parfois rendu impossible.

Principe de l'architecture

Si l'on observe la structure d'un système Linux, on s'aperçoit qu'une grande majorité du système de fichier peut être configuré en lecture seule. Mises à part quelques exceptions que nous décrirons plus loin, les parties nécessitant l'accès en lecture écriture sont les suivantes.

- Le répertoire `/var` contenant des données de fonctionnement souvent volatiles

- Le répertoire `/tmp` encore plus volatile!
- Les répertoires de configuration (exemple: définition d'adresse IP, etc.).
- Les répertoire d'utilisateurs ou applicatifs (exemple: stockage de données enregistrées par le système). Ces derniers pourront être placés sur un véritable disque dur et il est également possible de monter la partition à la demande afin de limiter les risques.

Outre la dernière catégorie que nous ne traiterons pas ici, il est donc relativement simple de mettre en place l'architecture suivante:

- La majorité du système est sur une partition en lecture seule.
- Le répertoire `/var` (point de montage) est sur un disque mémoire.
- Le répertoire `/tmp` est un lien symbolique sur `/var/tmp`.
- Le répertoire de configuration utilisera un format classique (EXT2, EXT3 ou JFFS2). La partition est de très faible taille. On peut même imaginer de stocker cette configuration sur une partition non formatée (au format TAR directement écrit sur le fichier spécial de la partition) ce qui limite les risques de problèmes de système de fichier dus à la coupure d'alimentation.

Mise en place (implémentation)

Au niveau de l'implémentation nous supposerons que nous utilisons un DoC de 8 Mo. Cette configuration correspond au cas réel d'un routeur Wifi construit sur la base d'un PC de type PC Light basé sur un processeur VIA C3. Le principe est d'utiliser 3 partitions sur la FLASH.

- Un première partition `/dev/nftla1` correspondant à `/boot` et contenant le noyau et les composants de LILO. Il n'est pas nécessaire que cette partition soit montée lors du fonctionnement du système mais seulement lors de la mise à jour éventuelle du noyau depuis un environnement de développement. De ce fait, cette partition sera formatée en EXT2.
- Une deuxième partition `/dev/nftla2` contenant le système de fichier. Elle sera formatée en CRAMFS comme décrit plus loin.
- Une troisième partition `/dev/nftla2` contenant les quelques fichiers de configuration. Elle est de très petite taille et utilise en première approche un formatage EXT2.

Le fichier `/etc/fstab` décrivant les montages sur le système cible aura l'allure suivante.

```
bash-2.05# cat /etc/fstab
/dev/nftla2     /                       cramfs  defaults        1       1
/dev/nftla3     /data                   ext2    defaults        0       0
none            /dev/pts                devpts  mode=622        0       0
none            /proc                   proc    noauto          0       0
/dev/ram0       /var                    ext2    defaults        0       0
```

On note que la partition `/boot` n'est pas montée mais que nous avons une partition `/var` montée sur un disque mémoire `/dev/ram0`.

Pour la deuxième partition (système de fichier principal), nous utiliserons le système de fichier CRAMFS. Ce dernier est présent dans l'arborescence du noyau Linux standard. C'est un système de fichier compressé (gzip), limité à 256 Mo, chaque fichier étant limité à 16 Mo. Il existe une petite documentation dans les sources du noyau (répertoire `Documentation/filesystems/cramfs.txt`). Pour utiliser CRAMFS, il faut disposer du programme `mkcramfs`. La version livrée avec certaines distributions ne fonctionne pas forcément

très bien et il vaut mieux récupérer la version officielle disponible sur le site du projet soit http://sourceforge.net/projects/cramfs.

Pour construire une partition CRAMFS sur un support physique on devra tout d'abord créer l'image du répertoire au format CRAMFS.

```
# mkcramfs my_root my_root.img
Directory data: 14092 bytes
Everything: 4252 kilobytes
Super block: 76 bytes
CRC: 8647fdf8
```

On pourra ensuite copier l'image sur la partition par un simple `dd` ou un `cp`.

```
# dd < my_root.img > /dev/nftla2
```

On pourra également la monter sur le système pour vérification en utilisant l'interface *loopback*.

```
# mount -t cramfs -o loop root_dir.img /mnt/cramfs
```

Au niveau du noyau, il faudra bien entendu valider le support de CRAMFS en statique au niveau du menu *File systems* de la configuration du noyau. Puisque nous utilisons également un disque mémoire, il faudra valider le support *ramdisk* le menu *Block devices*. La taille par défaut de 4 Mo pour le disque mémoire est suffisante pour notre application. Les deux figures qui suivent indiquent les configurations à effectuer.

Figure 10: Validation du support CRAMFS

Figure 11: Validation du support RAMDISK

Si nous faisons référence aux différentes publications concernant la construction d'un système Linux embarqué, nous savons que le démarrage du système est divisé en 5 étapes.

1. Le démarrage du système par LILO (Linux LOader) ou un programme équivalent type GRUB

2. Le chargement du noyau

3. Le lancement par le noyau du processus `init` (soit `/sbin/init`)

4. lecture du fichier `/etc/inittab` par le processus `init`. Ce fichier contenant le nom du fichier de démarrage comme décrit ci-dessous.

   ```
   # System initialization (runs when system boots).
   si:S:sysinit:/etc/rc.d/rc.S
   ```

5. L'exécution du script ci-dessus

Sachant que le répertoire `/var` est placé dans un disque mémoire, il est nécessaire de construire l'arborescence de ce dernier à chaque démarrage du système. On aura donc dans le fichier `rc.S` les lignes suivantes.

```
...
# Create /var (EXT2 filesystem)
/sbin/mke2fs -vm0 /dev/ram0 4096

# mount file systems in fstab (and create an entry for /)
# Mount /proc first to avoid warning about /etc/mtab
mount /proc
/bin/mount -at nonfs

# Populate /var
mkdir -p /var/tmp /var/log /var/lock /var/run /var/spool /var/lib/dhcp /
var/run/dhcpc
mkdir -p /var/cron/tabs /var/www/html /var/spool/cron /var/spool/mail /var/ppp
chmod a+rwx /var/tmp
...
```

Outre la création de /var, on notera le lien symbolique de `/etc/mtab` vers `/proc/mounts`. Ce lien est nécessaire car `/etc` n'est pas accessible en écriture.

```
lrwxrwxrwx    1 root     root           12 Jun 23  2003 mtab -> /proc/mounts
```

De même on notera l'utilisation fréquente des liens symboliques afin de permettre à des fichiers créés au démarrage d'apparaître dans le système de fichier en lecture simple (voir exemple de `/etc/resolv.conf` ci-dessous). Les fichiers variables sont systématiquement placés dans le répertoire `/var/run`.

```
# ls -l /etc/resolv.conf
lrwxrwxrwx    1 root     root           20 Sep 24 07:53 /etc/resolv.conf -> /
var/run/resolv.conf
```

Concernant les fichiers de configuration, ils sont placés dans le répertoire /data associé à la troisième partition. L'arborescence est similaire à celle utilisée dans les précédentes publications (chapitre 5 de l'ouvrage *Linux embarqué*).

```
# ls -l /data/sysconfig/
total 6
-rw-r--r--    1 65534    nobody         41 Jun 23  2003 general
-rw-r--r--    1 65534    nobody        349 Sep 17 21:14 network
-rw-r--r--    1 root     root          174 Jun 23  2003 ppp
-rw-r--r--    1 root     root          150 Sep  6  2003 pppoe
-rw-r--r--    1 root     root           16 Jun 23  2003 syslogd.opts
-rw-r--r--    1 root     root          150 Jun 24  2003 wireless
```

Bibliographie

- L'article *Construction d'un système Linux embarqué* paru dans Linux Magazine en septembre 2000 disponible sur http://www.ficheux.com/articles/lmf/embedded.
- L'ouvrage *Linux embarqué* paru aux édition Eyrolles en octobre 2002 dont la présentation est accessible depuis http://www.ficheux.com.
- L'article *Linux embarqué* par Michel Stempin. Linux Magazine, Novembre 2001 disponible sur

http://docs.mandragor.org/files/Misc/GLFM/lm33/Linux_Embedded.html.

- Le site *Embedded Linux page* par Patrice Kadionik sur http://www.enseirb.fr/~kadionik/embedded/embeddedlinux.html
- Site de la société M-Systems sur http://mmm.m-sys.com