Flashmon : un outil de trace pour les accès à la mémoire flash NAND

Jalil Boukhobza, Ilyes Khetib, Pierre Olivier

RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

Flashmon : un outil de trace pour les accès à la mémoire flash NAND

Jalil Boukhobza[+*], Ilyes Khetib[*], Pierre Olivier[*]

Université Européenne de Bretagne, Université de Brest
[+]UMR 3192 Lab-STICC,
[*]Département informatique,
20 Av. Le Gorgeu - CS 93837, 29238 Brest Cedex 3
boukhobza@univ-brest.fr, ilyes.khetib@etudiant.univ-brest.fr, pierre.olivier@etudiant.univ-brest.fr

Résumé
Les mémoires flash deviennent une technologie incontournable dans le domaine de l'embarqué. Leur succès est dû, essentiellement, à leur taille, leur légèreté, leur résistance au choc et leur faible consommation énergétique. Concernant les performances, l'avantage majeur que représentent ces mémoires, comparées aux disques durs, est leur performance homogène en lecture (séquentielle ou aléatoire). En revanche, pour les écritures, elles souffrent d'une durée de vie réduite à cause du nombre très limité d'effacements pouvant être appliqué à leurs cellules. L'un des axes de recherche le plus important consiste à établir des politiques permettant de niveler l'usure sur toute la surface afin de maximiser sa durée de vie moyenne. Il apparaît donc très important d'avoir à disposition des outils permettant de tracer les accès à la mémoire flash. Ceci permet, d'une part, lors de la conception, de tester et de valider des solutions mettant en œuvre des mémoires flash, et d'autre part, après implémentation, de surveiller l'usure en cours de fonctionnement. Ce papier présente Flashmon, un outil flexible et configurable, permettant de tracer, en temps réel, les accès à la mémoire flash NAND embarquée. Flashmon a été développé dans le cadre d'un projet étudiant de Master 2[e] année et sera proposé pour intégration dans les noyaux Linux.

Mots-clés : mémoires flash, trace, modules, Linux embarqué, MTD

1. Introduction

Les mémoires flash NAND sont de plus en plus utilisées comme système de stockage secondaire. On les trouve, sous différentes formes, dans les lecteurs de musique mp3, les smartphones, les appareils photos numériques, etc. L'utilisation de ces mémoires ne se cantonne plus au domaine de l'embarqué mais tend à s'élargir vers le domaine du PC et des grandes masses de données. En effet, la chute continue du prix au bit ainsi que la compatibilité avec les systèmes de stockage traditionnels facilite leur intégration.
Les faibles performances en écriture constatées lors de l'intégration de ces mémoires [11] sont dues au nombre limité d'opérations d'effacement qui peuvent être réalisées sur un bloc pour le réécrire. En effet, chaque cellule ne peut en supporter qu'un nombre variant entre 10^4 et 10^6 [14]. Cette contrainte majeure, rend difficile (voire impossible) la modification des données sur place. Pour cette raison, lorsqu'un bloc est modifié, les données sont copiées sur un autre emplacement et le premier est alors invalidé. Le but de cette technique est d'éviter d'user les mêmes blocs en les réécrivant à des emplacements différents afin de palier la localité spatiale et temporelle supposée d'une grande partie des charges d'E/S. Il s'agit alors de disperser les écritures sur la totalité de la surface afin d'en niveler l'usure, permettant ainsi d'augmenter la durée de vie moyenne de chaque bloc. Cette technique s'appelle le « wear leveling »et est intégrée de différentes manières dans les systèmes actuels.
Linux est un système d'exploitation incontournable pour les PC grand public ainsi que dans le domaine du calcul haute performance depuis plus d'une décennie. Il en va de même, depuis peu de temps, pour les systèmes embarqués (smartphones, set top box, routeurs, tablettes, etc). En effet, en plus de sa performance, fiabilité, scalabilité et homogénéité, son mode de développement (communauté et support) et sa licence le rendent extrêmement pertinent et attractif. Linux s'est donc vu intégrer un support permettant de gérer le stockage embarqué de données qui est généralement de la mémoire flash NAND. Le « wear

leveling »dans ce type de système, peut être introduit, essentiellement, de deux manières différentes : (1) intégré dans le contrôleur de la mémoire flash comme dans les Compact Flash, carte SD, disque SSD, ou (2) effectué par le système de fichiers si toutefois la mémoire est dénuée de contrôleur de ce type. Dans les systèmes embarqués, il est généralement question de la deuxième possibilité et l'outil développé s'applique à ce dernier. À notre connaissance, aucun outil de trace, tel que Diskmon ou USBmon, n'a été développé pour les accès à la flash.
Cet article est organisé comme suit : une première partie présente le cadre du projet et un bref état de l'art sur les mémoires flash et sur leur intégration sous Linux. Après cela, nous décrirons le projet ainsi que sa réalisation avant de conclure.

2. Contexte et état de l'art

Nous introduisons, dans cette section, le contexte du projet Flashmon et les concepts de base des mémoires flash. Ensuite, nous décrirons, globalement, leur intégration dans l'architecture du noyau Linux.

2.1. Le cadre du projet Flashmon

Comme cité précédemment, Flashmon a été développé dans le cadre de l'unité d'enseignement (UE) « Systèmes d'exploitation pour l'embarqué »pour les étudiants en 2e année de master « Logiciels pour les Systèmes Embarqués »à l'Université de Bretagne Occidentale. Dans cette UE, sont passés en revue les différents services des systèmes d'exploitation à travers plusieurs exemples différents. Le système d'exploitation utilisé pour les projets et TP reste néanmoins Linux, pour plusieurs raisons dont : (1) l'homogénéité, permettant d'établir un lien direct entre les concepts et l'implémentation, (2) la documentation et la communauté assez large procurant un gain en autonomie important pour les étudiants, et (3) la distribution sous licence GPL, idéale pour le développement système.
Le projet consistait à développer un traceur d'accès à la mémoire flash permettant de visualiser, en temps réel, les opérations effectuées. Il a été demandé aux étudiants, après qu'ils aient compris le fonctionnement de la mémoire flash sous Linux, de choisir les événements à tracer. Il leur fallait aussi choisir parmi un certain nombre d'outils disponibles. Une fois ces choix faits, il s'agissait de développer le traceur et de l'interfacer avec un visualiseur graphique. Là encore, il fallait choisir la librairie/fonctions à utiliser.
À travers ce projet, les étudiants ont pu se pencher sur plusieurs parties du système dont la manipulation de la chaîne de développement pour Linux embarqué, le développement de modules, les traceurs et debogueurs, la structure des systèmes de fichiers (pour flash et virtuels /proc), les bibliothèques graphiques, exploration et compilation du code du noyau, interface espace noyau/espace utilisateur, etc.

2.2. Les mémoires flash

Les mémoires flash sont des mémoires non volatiles de type EEPROM. Il y en a principalement de deux types : 1) la flash NOR et 2) la flash NAND. La première supporte un accès aléatoire par bit, une faible densité et un coût supérieur à la NAND et est utilisée pour le stockage du code. La seconde est accédée par bloc, présente une forte densité et un coût moindre, elle est utilisée pour le stockage des données. Une mémoire flash est composée d'une ou plusieurs puces, chaque puce étant subdivisée en plusieurs plans. Un plan est composé d'un nombre donné de blocs, qui à leur tour sont divisés en pages (de 512 octets à 2 KO). Les anciennes versions de flash NAND contiennent plusieurs blocs dont chacun dispose de 32 pages de 512 octets, alors que les blocs des versions actuelles contiennent 64 pages de 2KO.
Trois types d'opérations sont supportées sur ces mémoires : la lecture, l'écriture et l'effacement. La lecture et l'écriture s'effectuent sur une page, tandis que l'effacement se fait sur un bloc complet.
Écrire une donnée requiert d'avoir une page libre disponible (bloc préalablement effacé). Si le système doit modifier une page auparavant écrite, il cherche une page libre et y écrit les données, tout en invalidant la page antérieurement occupée. L'opération d'effacement est effectuée de manière asynchrone. S'il n'y pas de pages libres disponibles, un ramasse-miettes est exécuté pour recycler les pages invalidées.

2.3. Gestion des mémoires flash sous Linux

2.3.1. Systèmes de fichiers dédiés

Les techniques de « wear leveling »et de ramasse-miettes peuvent bénéficier d'un support matériel au niveau du contrôleur de la mémoire via la FTL (File Translation Layer) comme dans les SSD et clés USB, ou implémentées en logiciel à travers un système de fichiers dédié comme YAFFS, UBIFS, JFFS, etc. Il est à noter que dans le cas de l'utilisation d'une FTL, un système de fichiers standard est utilisé.

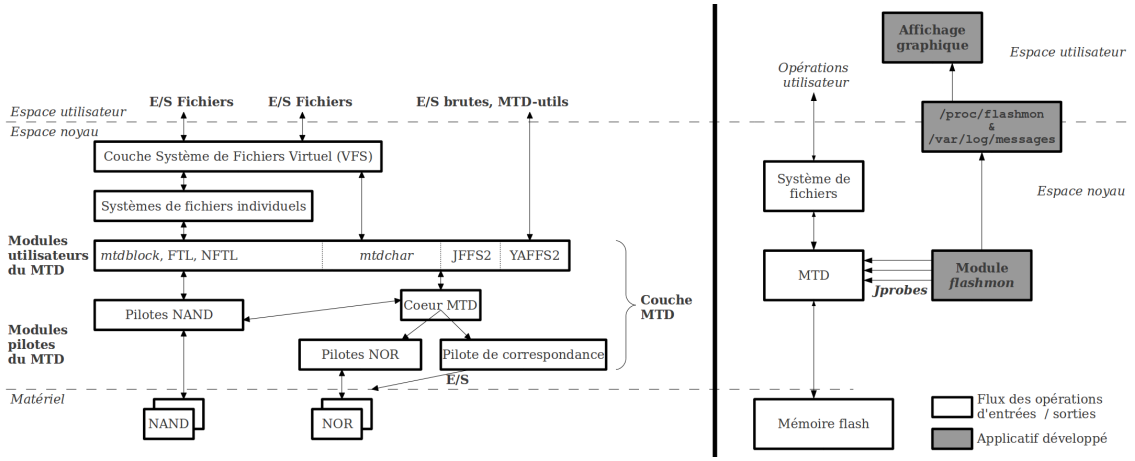

FIGURE 1 – *Schéma global de l'intégration du MTD à gauche et des outils développés à droite*

Dans le cas d'un système de fichiers dédié, Linux centralise les commandes vers la mémoire flash à travers une couche intermédiaire nommée *Memory Technology Device* [2]. C'est au travers de cette couche que le module écrit, dans le cadre de ce projet, récupère les informations sur les accès à la flash.

2.3.2. *Memory Technology Device*

Le MTD [2] est une couche logicielle qui interface les couches supérieures du système d'exploitation avec les pilotes des différentes mémoires flash embarquées comme le montre la figure 1. Le MTD étant une interface générique, son but est de rendre transparente l'utilisation de mémoires flash de différents types et de différents constructeurs. Le MTD a été développé, dans un premier temps, pour les mémoires NOR et a vu son utilisation étendue dans le cadre des mémoires NAND.

L'architecture du MTD se compose, en première approche, de quatre couches : 1) le cœur du MTD qui consiste en des librairies et structures utilisées par le reste du système, 2) les pilotes de bas niveau qui correspondent aux implémentations des mémoires NOR et NAND, 3) les pilotes de correspondances qui décrivent ce que le processeur doit faire en recevant des requêtes vers la flash et enfin 4) les applicatifs MTD qui sont, dans le cas qui nous intéresse, les systèmes de fichiers spécifiques (JFFS2 [1] par exemple).

Les périphériques MTD sont exportés en deux modes, caractère (entrée Mtdx dans /dev) et bloc (entrée MtdBlockx dans le /dev), vers l'espace utilisateur. Mtdx reçoit les commandes flash alors que MtdBlockx sert au montage/démontage du périphérique.

C'est au niveau du MTD que l'on récupère les informations sur les accès à la flash. Se faisant, on s'assure, d'une part, que l'on peut récupérer des informations quel que soit le système de fichiers utilisé, d'autre part que l'on ne risque pas de récupérer des informations erronées sur l'effacement de blocs logiques, et enfin, cela nous permet de rester indépendant vis à vis des pilotes de périphériques utilisés.

Nous avons retenu les structures et primitives contenues dans les fichiers suivants (ici `$LINUX` représente la racine des sources du noyau) [4] :

- `$LINUX/include/linux/mtd/mtd.h` : entête contenant les prototypes des fonctions, ainsi que les définitions de structures d'objets utilisés par le MTD, pour représenter des périphériques flash ;
- `$LINUX/include/linux/mtd/nand.h` : entête dédié à la manipulation des mémoires flash *NAND* ;
- `$LINUX/drivers/mtd/nand/nand_base.c` : primitives dont les prototypes sont dans `nand.h`.

3. L'architecture de Flashmon

Dans cette partie, sera présentée l'architecture globale de la solution proposée pour l'implémentation du traceur Flashmon. La figure 1 montre les différents outils développés et leur positionnement.

Après avoir explicité les événements tracés, nous décrirons les différentes contributions de Flashmon :

- un module pour le noyau Linux, s'appuyant sur des *Jprobes* [10] pour la trace d'événements ;
- des techniques de communication et de synchronisation mises en place entre le module et l'espace utilisateur (/proc et log de trace) ;

– une interface graphique s'appuyant sur la librairie graphique SDL [6].
Les traces d'exécution exposées dans cette section ont été collectées sur une carte embarquée *Armadeus* [7] APF27[8], munie de 256M octets de mémoire flash NAND. Pour ce qui est de la visualisation graphique, nous avons utilisé un écran LCD *Chimei* LW700AT [9].

3.1. Les événements tracés

Flashmon peut tracer 3 types d'évènements différents : la *lecture*, l'*écriture* de pages, et l'*effacements* de bloc. Le nombre de pages pouvant être très important, on regroupe les lectures / écritures par bloc. Il est à noter que Flashmon peut tracer un seul ou l'ensemble des types d'accès.
Les appels systèmes correspondant aux différents évènements flash à tracer sont les suivants :

```
/* Effacement d'un bloc : */
static int nand_erase(struct mtd_info *mtd, struct erase_info *instr);
/* Lecture flash : */
static int nand_read(struct mtd_info *mtd, loff_t from, size_t len,
        size_t *retlen, uint8_t *buf);
/* Ecriture flash : */
static int nand_write(struct mtd_info *mtd, loff_t to, size_t len,
        size_t *retlen, const uint8_t *buf);
```

Les définitions de ces fonctions se trouvent dans les sources du noyau Linux [3], au niveau du fichier suivant : `$LINUX/drivers/mtd/nand/nand_base.c`. Ce sont des appels systèmes relatifs à la couche *MTD* [2]. Ainsi, on s'assure que le module fonctionne quel que soit le système de fichiers utilisé en amont. Ces fonctions sont appelées par le système de fichiers et font, elles-mêmes, appel à des fonctions sous-jacentes, par exemple l'écriture de pages avec codes correcteurs d'erreurs.

3.2. Le module Flashmon

3.2.1. Modules et *Jprobes*

Un module est une portion de code pouvant être insérée à la volée dans le noyau. L'un des avantages de l'utilisation de cette méthode est qu'elle ne nécessite ni recompilation du code du noyau, ni redémarrage du système. De plus, mis à par le module, aucun programme supplémentaire n'est installé. D'autres outils auraient pu être utilisés tel que SystemTap [12], mais nous avons choisis de ne pas nous appuyer sur un outil qu'il faudrait installer, même si cela en rend le développement plus aisé.
Pour sonder les fonctions auxquelles on s'intéresse, on utilise des *Kprobes* [3], et plus particulièrement des *Jprobes* : ce sont des sondes que l'on pose sur une primitive système donnée. On associe à chaque sonde un *handler* ; une fonction exécutée à chaque appel à la primitive système sondée .
L'avantage des *Jprobes* est qu'elles donnent accès, au niveau du *handler*, aux paramètres de la fonction système sur laquelle elles sont posées. On peut, ainsi, récupérer plus de détails sur les accès tracés.

3.2.2. Fonctionnement du module

Le module maintient 3 tableaux de tailles égales au nombre de blocs de la partition flash tracée. Chaque élément du tableau est un compteur : un pour chaque type d'accès (lectures, écritures et effacements).
La fonction système `nand_erase` est appelée à chaque effacement de bloc. Ainsi, lorsque la sonde posée sur la fonction d'effacement détecte un appel, la case correspondante du tableau des effacements est incrémentée. On obtient le numéro du bloc effacé grâce à la structure `erase_info` passée en paramètre à `nand_erase` et récupérée par le *handler*.
Les fonctions de lecture / écriture flash `nand_read` et `nand_write` adressent des plages de données. Au sein du *handler*, on calcule alors les numéros de pages concernées par les requêtes, et les numéros des blocs contenant ces pages. On peut alors incrémenter les compteurs correspondants.
Pour le fonctionnement du module, on a besoin de données telles que le nombre de blocs total, le nombre de pages par bloc ou encore la taille d'une page en octets. Ainsi, au lancement du module on passe en paramètre l'indice de la partition flash à tracer (i.e. 4 pour `/dev/mtd4`) afin de récupérer les informations d'initialisation correspondantes.

3.3. Sorties et communication avec l'espace utilisateur

Comme on peut le voir dans la Figure 1, Flashmon produit deux types de sorties dans l'espace utilisateur. Une première, dite architecturale, qui donne l'état de la mémoire flash en nombre d'accès de différents types, et une deuxième, dite événementielle, dans laquelle les événements sont tracés selon l'instant auquel ils se sont produits.

3.3.1. Sorties architecturale et événementielle

Les événements d'entrée/sortie sont sauvegardés de deux manières différentes :

– Le répertoire `/proc`, est un système de fichiers virtuel permettant d'accéder directement à de nombreuses informations du noyau [13]. L'entrée *Flashmon* créée dans ce dernier, contient une cartographie de la mémoire flash. Chaque ligne correspond à un bloc et représente, dans l'ordre, le nombre de lectures, d'écritures et d'effacements qui lui sont associés :

```
2 0 1  // ligne x : bloc x ; 2 lectures 0 écritures 1 effacement
0 1 0  // ligne x+1 : bloc x+1 ; 0 lectures 1 écriture 0 effacement
```

– Le module produit des entrées dans le fichier journal du système `/var/log/messages` via le démon *syslogd* : la date de l'entrée/sortie, le type d'entrée/sortie, l'adresse du bloc (voir ci-dessous) :

```
Sep 9 05:55:18 arma user.debug kernel: Flashmon : WRITE page 28982 (block 452)
Sep 9 05:55:18 arma user.debug kernel: Flashmon : WRITE page 28981 (block 452)
Sep 9 05:55:19 arma user.debug kernel: Flashmon : ERASE block 237
```

3.3.2. Gestion des évènements tracés

La communication entre le module et l'espace utilisateur, et plus particulièrement l'application d'affichage, se fait par interruption logicielle. Les moyens utilisés sont les signaux. Ce mode de communication est asynchrone et de ce fait, dès qu'un événement de lecture, écriture ou effacement se produit sur la mémoire flash, un signal est envoyé vers un processus de l'espace utilisateur chargé de l'affichage.

En le recevant, l'application de visualisation met à jour les informations affichées. Ce mode de communication nous épargne une scrutation en continu de l'entrée `/proc/flashmon`. Un minimum de ressources est alors nécessaire (temps processeur).

3.4. Affichage graphique

En plus des sorties dans le /proc et dans le fichier log, nous avons développé une interface graphique permettant une visualisation plus claire et plus directe des accès à la flash en temps réel. Cela permet de visualiser sur des écrans tels que l'écran LCD *Chimei* LW700AT [9]. Les solutions possibles permettant de produire un affichage graphique sont diverses, parmi ces dernières :

– Le *Framebuffer* [4] : Cette solution ne nécessite aucune librairie vu que l'affichage se fait directement en écrivant des informations (valeur d'un pixel) dans la zone mémoire du framebuffer. L'inconvénient est la nécessité d'implémenter des primitives graphiques.
– *DirectFB* [5] : Cette solution est trop lourde en terme de fonctionnalités offertes par rapport à ce que l'on souhaite utiliser pour l'affichage graphique. Elle est également coûteuse en ressources matérielles.
– *SDL* [6] : La librairie SDL est à mi-chemin entre les deux solutions précédentes. La librairie de base (incluse lors de la construction du système de fichiers) répond amplement à nos besoins d'affichage. SDL est simple à utiliser et est peu gourmande en ressources matérielles. Elle offre également la possibilité d'utiliser le clavier pour dérouler la fenêtre d'affichage.

L'application graphique développée affiche à l'écran une matrice qui représente la mémoire flash en temps réel (voir la figure 2). Un curseur entourant un bloc donné de la matrice peut être déplacé à l'aide du clavier afin d'afficher les informations relatives à ce bloc : son numéro, le nombre d'entrées/sorties effectuées selon l'accès choisi (lectures, écritures, effacements). La visualisation est aussi facilitée par le fait que plus un bloc est écrit, plus sa couleur fonce. Cela permet d'avoir rapidement une idée de l'état des différents blocs les uns par rapport aux autres.

3.5. Gestion du fichier de logs

En fonction du temps de trace, le nombre d'opérations de lectures, écritures et effacements sur la mémoire flash peut vite devenir très important. Le fichier système `/var/log/messages` est mis à jour d'une façon continue et sa taille augmente considérablement, il peut, par conséquent, saturer la partition sur laquelle il se trouve. Afin d'éviter ce problème, différentes solutions peuvent être envisagées :

– utiliser la rotation de log : la compression par exemple, ce qui permet un gain d'espace ;
– rediriger le log sur le réseau (utilisation d'un serveur de log) pour éviter une saturation de la flash.

4. Conclusion et perspectives

Nous avons réalisé un outil de trace des accès à la mémoire flash permettant, d'une part, lors de la conception, de tester et de valider des solutions telles que des systèmes de fichiers pour mémoires flash,

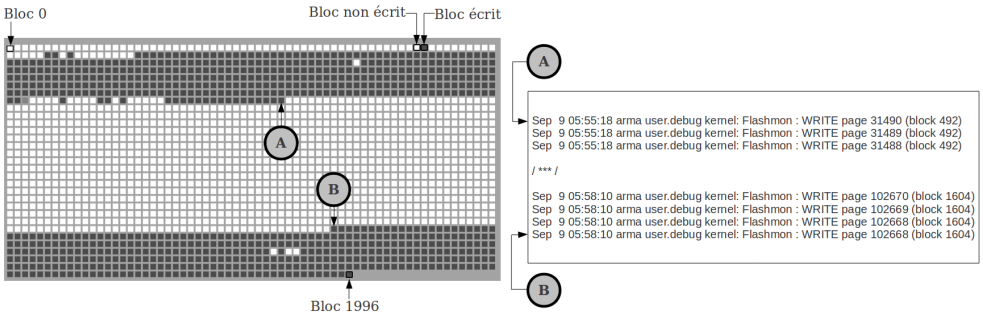

FIGURE 2 – *Capture d'écran des écritures, et correspondance dans le log du noyau* : à gauche, est visualisé l'affichage après la copie d'un fichier de 70 MOs. Les parties sombres représentent les blocs écrits. À droite, est montrée la correspondance avec le log. Les blocs (A) et (B) représentent respectivement le début et la fin du fichier écrit. On voit que JFFS2 écrit les données dans le sens inverse de la numérotation des blocs. Cette figure illustre bien la vision par JFFS2 de la flash comme un *log circulaire* [1].

et d'autre part, après implémentation, de surveiller l'usure en cours de fonctionnement d'un certain nombre d'applicatifs. Le but de l'usage d'un tel outil peut être double : l'optimisation de la durée de vie en surveillant les effacements, ou l'optimisation de performance en recherchant des patrons d'accès aux fichiers spécifiques au comportement/paramètre d'une mémoire flash donnée. Le travail sera étendue en intégrant la possibilité d'envoyer les logs sur le réseau pour rendre l'outil le moins intrusif possible. Une autre piste serait de récupérer, pendant l'initialisation, l'état d'effacements de la flash plutôt que de commencer le comptage à zéro. Nous sommes aussi en train de développer un outil permettant de traduire les traces de Flashmon pour qu'elles soient rejouées dans le simulateur de systèmes de stockage Disksim. Flashmon pourra aussi être facilement intégré à des systèmes se basant sur le noyau Linux tel que Android et à tout autre systèmes unixien (utilisant une MTD). Le projet a été effectué dans le cadre d'une unité d'enseignement, cela a permis aux étudiants d'appréhender plusieurs des éléments clés de la programmation système. Nous sommes parvenus à travers ce projet à coupler pédagogie et enseignement avec une réponse à un besoin actuel en termes d'outil pour le monde académique et industriel. L'outil sera proposé en licence GPL.

Bibliographie

1. JFFS2 : The Journalling Flash File System, version 2, `http://sourceware.org/jffs2/`, acc .le 6 dec. 2010
2. Memory Technology Device (MTD) Subsystem for Linux, `http://www.linux-mtd.infradead.org/`, acc. le 6 dec. 2010
3. The Linux Kernel Archives, `http://www.kernel.org/`, acc. le 7 dec. 2010
4. Framebuffer, `http://en.wikipedia.org/wiki/Framebuffer/`, acc. le 7 dec. 2010
5. DirectFB, `http://directfb.org/`, acc. le 7 dec. 2010
6. SDL, `http://www.libsdl.org/`, acc. le 7 dec. 2010
7. Armadeus Project, `http://www.armadeus.com/wiki/index.php?title=Main_Page`, acc. le 22 jan. 2011
8. Armadeus APF27 `http://www.armadeus.com/wiki/index.php?title=APF27`, acc. le 22 jan. 2011
9. Chimei LW700 `http://www.armadeus.com/wiki/index.php?title=Chimey_LW700`, acc. le 22 jan. 2011
10. Keniston (Jim), Panchamukhi (Prasanna S.) et Hiramatsu (Masami) – "Kernel Probes", `http://www.mjmwired.net/kernel/Documentation/kprobes.txt`, acc. le 31 jan. 2011
11. Gupta (A.), Kim (Y.), Urgaonkar (B.) – DFTL : A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings– ACM ASPLOS, Washington, 2009.
12. Frank Ch. Eigler et al. – Architecture of systemtap : a Linux trace/probe tool–, 2005
13. Olivier Daudel – /proc et /sys – Editions O'Reilly, 2006
14. Chen (F.), Koufaty (D. A.), Zhang (X.), –Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives– ACM SIGMETRICS, 2009.