18 jan 2009

FUSE, développez vos systèmes de fichiers dans l'espace utilisateur

Catégorie : Administration système Tags : GLMF

~~Retrouvez cet article dans : Linux Magazine 92~~

Comment décrire FUSE succinctement ? Hum ! On peut déjà dire ce qu'il n'est pas : FUSE n'est pas un émulateur du micro-ordinateur Spectrum. De la même façon, FUSE n'est pas un satellite d'observation de la NASA… ni même le titre d'un album rock de l'année 69 ou encore celui d'un film de nationalité bosniaque. Enfin, oui, c'est aussi tout cela (c'est fou comme les mots de 4 lettres peuvent être tellement de choses), mais en ce qui nous concerne, FUSE est surtout et avant tout une extension au noyau Linux autorisant la création de systèmes de fichiers s'exécutant dans l'espace utilisateur.

PREAMBULE

Car, qui n'a jamais rêvé de pouvoir implémenter toutes les fonctionnalités d'un système de fichiers dans l'espace utilisateur ?
En effet, écrire un système de fichiers nécessite en général le développement d'un driver et donc de lier son code à la destinée de celui du noyau. Entre autres désavantages, cela signifie restreindre, pour le plus grand nombre, la diffusion de votre système de fichiers aux nouvelles variantes du noyau Linux ;

ce qui n'est sans doute pas la meilleure solution pour viser un public le plus large possible.
Exercice de style en soi particulièrement formateur, l'écriture d'un driver, même celui d'un système de fichiers, n'est pas une chose aisée. Outre les connaissances spécifiques nécessaires à ce type de développement, cela occupe pas mal de temps et épuise rapidement un stock non négligeable de café noir1.
Pour enfoncer le clou, accéder à l'espace réservé du noyau est un exercice périlleux ; au moindre comportement inattendu de votre driver, le risque n'est pas nul de mettre en péril toute la stabilité du système d'exploitation.
Pour les systèmes de fichiers classiques qui lisent et écrivent des données sur des supports physiques, cela est difficilement contournable ; pour optimiser les performances, il est indispensable de se trouver au cœur des débats.
Mais pour les systèmes de fichiers dits « virtuels », qui ne manipulent pas directement les données, mais servent souvent d'interface avec des données abstraites ou distantes, on se prend à rêver d'une solution qui nous permettrait de nous soustraire de la nécessité impérieuse de passer par le noyau2.
Et c'est justement de cette sacro-sainte dépendance au noyau que FUSE [1] se propose de nous soulager ; FUSE étant l'acronyme de « Filesystem in USErspace », soit « Système de fichiers en espace utilisateur » pour les plus anglophobes d'entre vous.
FUSE a fait officiellement son apparition dans le noyau Linux à partir de la version 2.6.14 et se compose d'un module noyau que l'on accède par l'intermédiaire du fichier spécial /dev/fuse et d'une bibliothèque en espace utilisateur libfuse.
FUSE a été dérivé de la bibliothèque AVFS [2] (A Virtual FileSystem) qui visait le développement d'une interface permettant à n'importe quel programme de lire des données présentes sur des disquettes, dans des fichiers compressés gzip, tar, zip, bzip2, ar et rar2 ou des protocoles distants comme ftp, http, webdav, rsh/rcp et ssh/scp… cela sans recompiler son application (en somme, gnome-vfs sous GNOME ou kioslaves sous KDE, mais accessible depuis n'importe quelle application).

AVFS :
Techniquement, le projet valorise avant exécution du programme cible la variable d'environnement ~~LD_PRELOAD~~ qui permet de surcharger sélectivement les fonctions des autres bibliothèques partagées et donc de rediriger les appels système d'entrées-sorties vers ceux d'AVFS.

Depuis, le projet s'est scindé en deux parties distinctes, pour notre plus grand bonheur, en libérant la partie la plus prometteuse, à savoir FUSE !
Sous Linux, il existe une version 2.4.x qui peut être compilée sur les noyaux 2.4 et une version 2.6.x qui peut être compilée sur les noyaux 2.6. La dernière

version en date est la 2.6.1 (sortie le 1er décembre 2006).
Coté licence, le module noyau ainsi que les utilitaires associés sont placés sous licence GPL, alors que la bibliothèque ~~libfuse~~ (liée dynamiquement avec les applications) est sous licence LGPL (ce qui autorise tout naturellement l'utilisation de FUSE dans des applications non GPL).
Pour information, Linux n'est pas le seul système d'exploitation à disposer de FUSE (bien que ce soit la plate-forme native) : un portage est heureusement disponible pour FreeBSD 6.x et 7.x. [3] ainsi que pour Mac OS X [4] (à noter que pour ce dernier, c'est GOOGLE qui a réalisé le portage). Il va sans dire que puisque chaque implémentation respecte la même interface, tous les systèmes de fichiers virtuels reposant sur FUSE fonctionnent sur n'importe laquelle des trois plateformes (moyennant une compilation native des exécutables). Cela étend le nombre d'utilisateurs potentiels pour chacune de ces applications et permet d'ores et déjà de profiter d'un nombre respectable de systèmes de fichiers virtuels respectant les interfaces FUSE.
Car, puisque cela fonctionne dans l'espace utilisateur, tout peut être transformé en système de fichiers. Cela va de l'accès en lecture/écriture au système de fichiers NTFS, à l'accès distant par SSH ou encore à des systèmes de fichiers plus improbables, comme la translation temps-réel de fichiers au format FLAC ou OGG vers le format MP3 (Yacufs), l'accès transparent à une base de données MySQL (MySQLFS) ou encore l'accès aux articles Wikipédia comme de simples fichiers (WikipediaFS) [5]...
Tout peut être transposable vers FUSE ; cela ouvre des horizons nouveaux uniquement limités par l'imagination des développeurs (et beaucoup s'accordent à dire que l'on en est qu'au début).
La facilité de développement d'un système de fichiers FUSE est étonnante de simplicité du moment que l'on en a intégré les principes. Le développement en espace utilisateur permet un développement plus aisé et facilite grandement les tests ; et cela sans endommager l'intégrité du système d'exploitation ou altérer ses données.
L'un des seuls défauts de cette solution est que les performances sont forcément moindres tout en restant raisonnables par rapport à une solution purement noyau ; en réalité, la différence est peu perceptible, vu de l'utilisateur, si l'on considère le type de systèmes de fichiers virtuels qui est visé par cette solution.
Plus généralement, FUSE est à mettre au même plan que le chargement dynamique des modules dans le noyau Linux. C'est une solution qui permet d'étendre à chaud les fonctionnalités de ce dernier (cf. rajouter de nouveaux systèmes de fichiers) et représente en cela une réelle évolution plus qu'une révolution.
Certains rappelleront à juste titre que le Hurd fait cela bien mieux par l'intermédiaire des translators, et d'autres que FUSE est un hack assez malin autour du noyau, mais que cela reste au final un hack. A fonctionnalités égales, il faut reconnaître que les deux solutions se valent, mais peut-être pas avec la

même élégance.
Pour suspendre temporairement ce débat, on me souffle à mon oreillette qu'une version Hurd serait en cours de développement (en version vraiment préliminaire)… de même qu'un portage serait en cours pour OpenSolaris[6].
Quoi qu'il en soit, il faut retenir que FUSE nous évite de tout développer dans le noyau et permet que notre code soit plus facile à implémenter, plus facile à déployer et plus facile à tester.

INSTALLATION DE FUSE

Première étape indispensable avant d'essayer un système de fichiers FUSE : vérifier que le module noyau est présent sur votre distribution. Bien que celui-ci ne soit disponible en standard que depuis le 2.6.14, certaines distributions l'ont déjà intégré dans des versions antérieures. C'est le cas de la SUSE 10.0 qui fournit le module dans sa version maison du noyau 2.6.13.

```
# /sbin/modprobe -l fuse
/lib/modules/2.6.13-15.12-default/kernel/fs/fuse/fuse.ko
```

Sinon, il vous reste à compiler le module et à l'installer sur votre système. L'opération n'est pas compliquée et fait appel au désormais récurent « ~~./configure && make && make install~~ » (cela va aussi installer de surcroît la bibliothèque et un utilitaire système).
Si le module est présent dans votre noyau Linux, la seconde étape va consister à installer les paquets contenant la bibliothèque ~~libfuse~~ et l'utilitaire ~~fusermount~~ nécessaires pour démonter les points de montage FUSE (étape inutile en cas d'installation à partir des sources).
Sur la SUSE, Fedora ou Mandriva, il vous faudra installer les paquets ~~fuse~~ et ~~fuse-devel~~ ; sur une Debian ou ses dérivées (Ubuntu, Kubuntu…), il vous faudra installer les paquets ~~libfuse2~~, ~~libfuse-dev~~ et ~~fuse-utils~~.
Si vous avez installé les sources, il existe un moyen assez simple d'interroger la version de l'~~API~~ de la bibliothèque libfuse (la dernière en date étant la 2.6 ; on peut suivre l'historique sur [7]) :

```
 # more /usr/include/fuse/fuse_common.h
/** Major version of FUSE library interface */
#define FUSE_MAJOR_VERSION 2
/** Minor version of FUSE library interface */
#define FUSE_MINOR_VERSION 5
```

La troisième et dernière étape consiste à rendre le fichier spécial ~~/dev/fuse disponible~~ pour toutes vos applications en lecture et écriture.
Pour charger directement le driver, un simple ~~modprobe~~ fera l'affaire (sous ~~root~~) :

```
# modprobe fuse
# lsmod  | grep fuse
fuse                   33420  0
```

Pour connaître la version de l'API du module, vous pouvez consulter le tampon des messages du noyau :

```
# dmesg | grep fuse
fuse init (API version 7.6)
```

Pour rendre le chargement du driver persistant, ajoutez le nom du module dans le fichier ~~/etc/modules~~ sur Debian et ses dérivés :

```
$ echo fuse >> /etc/modules
```

Sur la SUSE, il sera nécessaire de rajouter le nom du module dans la variable ~~MODULES_LOADED_ON_BOOT~~ du fichier ~~/etc/sysconfig/kernel~~ pour forcer le chargement du driver à l'amorçage du système :

```
# grep MODULES_LOADED_ON_BOOT /etc/sysconfig/kernel
MODULES_LOADED_ON_BOOT=»fuse»
```

Lorsque le module ~~fuse~~ est chargé, il n'est pas pour autant accessible par tous les utilisateurs pour une question de droits (sur la Debian, il faut que l'utilisateur appartienne au groupe ~~fuse~~) ce qui est un comble pour une fonctionnalité qui est censée être accessible par tous les utilisateurs (car, comme on le verra plus loin, l'aspect sécurité est pris en compte par FUSE).

On va corriger cela en modifiant les droits du fichier spécial en jouant sur les paramètres UDEV :

```
$ cat /etc/udev/rules.d/40-fuse.rules
KERNEL==»fuse», NAME=»%k», MODE=»0666»
```

Il sera peut-être même nécessaire de changer les droits de l'utilitaire Fusermount :

```
$ chmod 4755 /usr/bin/fusermount
```

MISE EN PRATIQUE

Vous êtes fin prêt pour les tests pratiques.
Nous allons monter une image ~~iso~~ en lecture seule en s'appuyant sur l'application ~~fuseiso~~ que vous pourrez télécharger sur [8].
En effet, à moins que celle-ci ne soit disponible sous forme binaire pour votre distribution, il est nécessaire de l'installer à partir des sources :

```
$ tar xjvf fuseiso-20061017.tar.bz2
$ cd fuseiso-20061017/
$ ./configure
$ make
$ make install
```

Si vous ne disposez pas d'une image iso, vous pouvez en créer une par la commande :

Il reste alors à monter l'image iso sur un répertoire vide (de préférence), comme vous le feriez avec la commande mount :

```
$ mkdir -p /tmp/mount
$ fuseiso -n /tmp/test.iso /tmp/mount/
$ mount | grep fuse
fuseiso on /tmp/mount type fuse (rw,nosuid,nodev)
$ ls /tmp/mount
...
```

Démontez le système de fichier fait appel à la commande fusermount (la commande umount ne fonctionnera pas) :

```
 $ fusermount  -u /tmp/mount/
```

Voilà, ce n'est pas plus compliqué que ça. N'importe quel utilisateur peut à sa convenance monter et démonter une image iso !
Il est même possible de monter automatiquement un tel système de fichiers dans /etc/fstab en utilisant le type de système de fichiers FUSE (par la magie du script /sbin/mount.fuse) :

```
fuseiso#/tmp/test.iso   /tmp/mount fuse user 0 0
```

ET LA SECURITE DANS TOUT CA ?

Plusieurs mécanismes sont mis en œuvre pour améliorer la sécurité et faire que FUSE ne devienne trop perméable aux attaques :

- Par défaut, seul l'utilisateur ayant effectué un montage peux accéder ou démonter ce point de montage (même le super-utilisateur en est exclu !). Cela sécurise l'accès aux données.
- Les options nosuid et nodev sont forcées par défaut lors du montage. Il n'est pas possible de positionner un bit suid pour prendre l'identité de quelqu'un d'autre de même que les fichiers spéciaux ne sont pas interprétés sur ce type de système de fichiers.

Pour autoriser tous les utilisateurs à accéder au point de montage, il faut ajouter l'option allow_other lors de l'appel à notre application :

```
fuseiso -n /tmp/test.iso /tmp/mount/ -o allow_other
```

Par symétrie, autoriser l'accès en super-utilisateur nécessite l'utilisation de l'option allow_root sachant que ces deux paramètres ne sont pris en compte que lorsque le fichier /etc/fuse.conf est valorisé avec l'option user_allow_other. Le fait que l'utilitaire fusermount possède un suid-bit positionné à root peut difficilement être contourné, mais, finalement, comme c'est exactement la même chose pour l'utilitaire mount, ce n'est pas en soi un véritable problème, car lié à la conception même du noyau Linux.

A PROPOS, COMMENT CA MARCHE ?

L'API native est écrite en C avec de nombreux binding vers le C++, le Java, le C#, ainsi que des langages scripts tels que Python, Perl, TCL ou Ruby (cette liste n'est pas exhaustive).
Prenons un exemple pédagogique. Imaginez que nous souhaitions implémenter un module FUSE qui présente sous forme de fichiers tous les utilisateurs du système avec, comme contenu, la description des utilisateurs (dans cette optique, il nous vient tout naturellement à l'esprit d'utiliser la base de données que constitue le fichier /etc/passwd).
Pour cela, notre programme, écrit en langage C et réduit à sa plus simple expression pour faciliter la lecture, va implémenter les appels système readdir, open et read. Il faut rajouter à cela l'appel getattr qui n'existe pas dans la libc et qui doit être vu comme l'équivalent de l'appel système stat. Cette fonction occupe une place toute particulière dans FUSE, car elle sera appelée avant tout accès à un chemin dans le système de fichiers (il est alors possible de centraliser pas mal de traitements dans cette fonction suivant le type de système de fichiers que l'on implémente).
Toutes les fonctions sont alors regroupées dans une structure de type fuse_operations, puis passées en paramètre à la fonction bloquante fuse_main(), laquelle va gérer tous les évènements sur le système de fichiers en collaboration avec le driver fuse (figure 1).
Comme précisé dans la figure 1, ces appels système vont remplacer ceux de la bibliothèque C standard par l'intermédiaire de la bibliothèque libfuse (elle-même interfacée avec le fichier spécial /dev/fuse).
Voilà comment il est possible d'implémenter un tel système de fichiers sous FUSE :

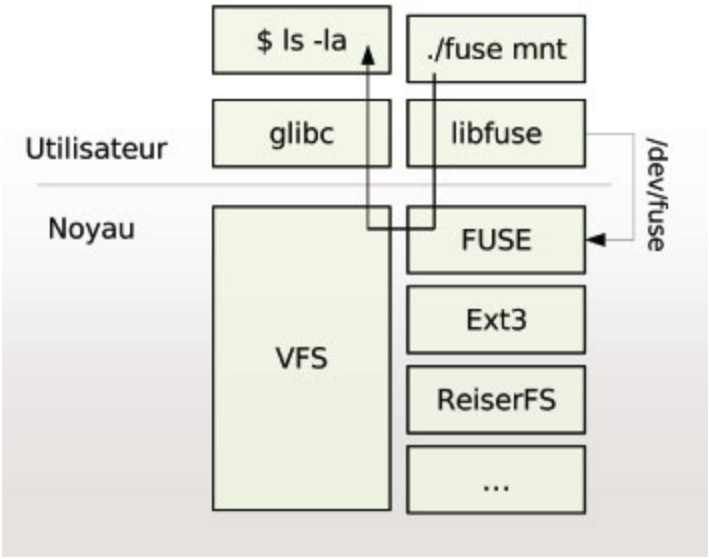

Figure1 : Interaction avec le driver fuse et la libc

```
#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <pwd.h>
struct passwd *pw;
static int passwd_getattr(const char *path, struct stat *stbuf)
{
    memset(stbuf, 0, sizeof(struct stat));
    if(strcmp(path, "/") == 0) {
        stbuf->st_mode = S_IFDIR | 0755;
        stbuf->st_nlink = 2;
    return 0;
    }
    pw = getpwnam( &(path[1]) );
    if (pw == NULL) return -ENOENT;
    stbuf->st_mode = S_IFREG | 0444;
    stbuf->st_nlink = 1;
    stbuf->st_uid = getuid();
    stbuf->st_gid = getgid();
    stbuf->st_size = (off_t)strlen(pw->pw_gecos)+1;
    return 0;
}
static int passwd_readdir(const char *path, void *buf,
fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi)
{
    (void) offset;
    (void) fi;
    if(strcmp(path, «/») != 0) return -ENOENT;
    filler(buf, «.», NULL, 0);
```

```
    filler(buf, «..», NULL, 0);
    setpwent();
    while ((pw = getpwent())) filler(buf,pw->pw_name,NULL,0);
    endpwent();
    return 0;
}
static int passwd_open(const char *path, struct fuse_file_info *fi)
{
    pw = getpwnam( &(path[1]) );
    if (pw == NULL) return -ENOENT;
    if ((fi->flags & 3) != O_RDONLY)
        return -EACCES;
    return 0;
}

static int passwd_read(const char *path, char *buf, size_t size,
off_t offset, struct fuse_file_info *fi)
{
    size_t len;
    (void) fi;
    pw = getpwnam( &(path[1]) );
    if (pw == NULL) return -ENOENT;
    len = (size_t)strlen(pw->pw_gecos)+1;
    if (offset < len) {
        if (offset+size > len) size=len-offset;
        memcpy( buf, (pw->pw_gecos)+offset, size );
        buf[size-1] = '\n';
        return size;
    }
    return 0;
}

static struct fuse_operations passwd_oper = {
    .getattr    = passwd_getattr,
    .readdir    = passwd_readdir,
    .open       = passwd_open,
    .read       = passwd_read,
};

int main(int argc, char *argv[])
{
    return fuse_main(argc, argv, &passwd_oper);
}
```

Hormis la spécificité de l'appel à la fonction fuse_main() avec le passage en argument de la liste des appels système qui seront utilisés par le système de fichiers (structure fuse_operations), rien n'est fondamentalement compliqué à appréhender dans ce programme pour quelqu'un qui manipule raisonnablement le langage C.
La liste et les prototypes des appels système qu'il est possible d'implémenter sont disponibles dans le fichier /usr/include/fuse/fuse.h. Je vous invite cordialement à y faire un petit tour.

Les puristes n'auront pas oublié de noter que, pour bien faire, il faudrait utiliser les versions ré-entrantes des primitives qui manipulent le fichier /etc/passwd. Cela a volontairement été omis dans le listing précédent par souci de clarté.
Pour générer l'exécutable, tapez la commande suivante dans une console :

```
$ gcc -Wall -o fuse fuse.c -lfuse
```

L'utilisation de l'option ~~-D_FILE_OFFSET_BITS=64~~ sera sans doute nécessaire pour compiler correctement FUSE. Cette option, utile au compilateur, permet de demander de remplacer de façon transparente les appels système 32 bits vers ceux 64 bits, et ce, même sur architecture 32 bits (par exemple, l'appel système stat64 sera renommé en ~~stat~~).

Si le programme ne veut toujours pas compiler, l'astuce consiste alors à préciser au compilateur quelle version de l'interface de programmation libfuse on souhaite utiliser en valorisant l'option -DFUSE_USE_VERSION=XX. Sur la machine de test, les versions 22, 25 et 26 ont donné de bon résultats.
Passons à la mise en pratique de notre programme :

```
$  mkdir mnt
$ ./fuse mnt/
$ cd mnt/
$ ls

at     bin         games   laurence   lionel    mail  wwwrun
man    messagebus  news    ntp        postfix   root  uucp
avahi  daemon      ftp     haldaemon  ldap      lp    vdr
mdnsd  mysql       nobody  nx         postgres  sshd

$ ls -la games
-r--r--r--  1 lionel users 14 1970-01-01 01:00 games
$ cat games
Games account
$ cd ../
$ fusermount -u mnt
```

A noter aussi que le programme (dont les sources sont disponibles sur le site [9]) peut être appelé avec de nombreuses options que l'on peut consulter avec le flag ~~-h~~. En particulier, l'option ~~-d~~ permet d'obtenir pas mal d'informations intéressantes sur le fonctionnement interne du programme qui sinon passe immédiatement en tâche de fond.

ET VOUS ? VOUS PRENDREZ QUOI COMME SYSTEMES DE FICHIERS ?

Référencés sur le site officiel, il existe actuellement un peu moins d'une centaine de systèmes de fichiers conçus au-dessus de FUSE. Dans la réalité, ils doivent être un peu plus nombreux.
Tous ne présentent peut-être pas le même intérêt pratique, et certains sont même d'inspirations douteuses, mais cela signifie surtout que cette technologie peu connue ouvre une fenêtre vers une multitude d'expérimentations en tout genre qui devrait nous offrir quelques belles pépites et étendre un peu plus la puissance et la modularité de notre OS favori.
Un petit passage en revue d'un florilège de système de fichiers [10] qui devraient vous donner envie de vous intéresser à cette technologie :

NOM	DESCRIPTION
SSHFS	Parcourir une arborescence distante par l'intermédiaire du protocole SFTP
BlogFS	Pour consulter les *blogs* WordPress comme de simples fichiers
Captive NTFS	Lire et écrire sur des partitions NTFS au travers du driver natif Windows
Cddfs	Monter vos CD-Rom audio pour accéder au contenu WAV de façon transparente
EncFS	Système de fichier crypté pour sécuriser vos données
FuseDAV	Parcourir les partages WebDAV
Fusecram	Monter les images compressées au format cramfs
FuseCompress	Compresser des données à la volée
GnomeVFS2	Monter tous les systèmes de fichiers accessibles par Nautilus sous GNOME
gphoto2-fuse-fs	Permet de d'accéder à un appareil photo numérique supporté par gPhoto2 sans passer par gtkam ou gPhoto2
Kio Fuse Gateway	Monter les *ioslaves* KDE pour les exposer aux autres applications
Mountlo	Monter des fichiers en *loopback*
WikipediaFS	Voir et éditer les articles Wikipedia comme de simples fichiers
Yacufs	Convertit à la volée tous vos fichiers `.ogg`, `.flac` en fichiers MP3

Un dernier mot avant de refermer cet article ? Le mot FUSE signifie « fusible » en anglais et c'est peut-être ce qui le caractérise le mieux ; la capacité de développer et d'ajouter de nouveaux systèmes de fichiers sous Linux sans mise à jour du noyau et, surtout, sans mettre en péril sa robustesse. À vos claviers !

Liens

- [1] http://fuse.sourceforge.net/
- [2] http://sourceforge.net/projects/avfs
- [3] http://fuse4bsd.creo.hu/
- [4] http://code.google.com/p/macfuse/
- [5] http://wikipediafs.sourceforge.net/
- [6] http://www.opensolaris.org/os/project/fuse/
- [7] http://fuse.sourceforge.net/wiki/index.php/ApiChangelog
- [8] http://freshmeat.net/projects/fuseiso/
- [9] http://lionel.tricon.free.fr/Articles/fuse/exemple
- [10] http://fuse.sourceforge.net/wiki/index.php/FileSystems

~~Retrouvez cet article dans : Linux Magazine 92~~

Posté par (La rédaction) | Signature : Lionel Tricon | Article paru dans

Laissez une réponse

Vous devez avoir ouvert une session pour écrire un commentaire.

« Précédent Aller au contenu »
Identifiez-vous
Inscription
S'abonner à UNIX Garden

Articles de 1ère page

- Git, les mains dans le cambouis
- PostgreSQL 8.3 : quoi de neuf ?
- Introduction à Ruby on Rails
- Linux Pratique HS N°17 - Mars/Avril 2009 - Chez votre marchand de journaux
- 88 miles à l'heure !
- Développement et mise en place d'un démon Unix
- Calculer ses rendus Blender en cluster ou comment faire sa propre «render farm» avec DrQueue

- Technologie rootkit sous Linux/Unix
- CMake : la relève dans la construction de projets
- Des petits sondages pour améliorer nos magazines

Catégories

- Administration réseau
- Administration système
- Agenda-Interview
- Audio-vidéo
- Bureautique
- Comprendre
- Distribution

- Embarqué
- Environnement de bureau
- Graphisme
- Jeux
- Matériel
- News
- Programmation
- Réfléchir
- Sécurité
- Utilitaires
- Web

Articles secondaires

- 30/10/2008
  Google Gears : les services de Google offline

  Lancé à l'occasion du Google Developer Day 2007 (le 31 mai dernier), Google Gears est une extension open source pour Firefox et Internet Explorer permettant de continuer à accéder à des services et applications Google, même si l'on est déconnecté....

  Voir l'article...
  7/8/2008
  Trois questions à...

  Alexis Nikichine, développeur chez IDM, la société qui a conçu l'interface et le moteur de recherche de l'EHM....

  Voir l'article...
  11/7/2008
  Protéger une page avec un mot de passe

  En général, le problème n'est pas de protéger une page, mais de protéger le répertoire qui la contient. Avec Apache, vous pouvez mettre un fichier ~~.htaccess~~ dans le répertoire à protéger....

  Voir l'article...
  6/7/2008
  hypermail : Conversion mbox vers HTML

  Comment conserver tous vos échanges de mails, ou du moins, tous vos mails reçus depuis des années ? mbox, maildir, texte... les formats ne manquent pas. ...

Voir l'article...
6/7/2008
iozone3 : Benchmark de disque

En fonction de l'utilisation de votre système, et dans bien des cas, les performances des disques et des systèmes de fichiers sont très importantes....

Voir l'article...
1/7/2008
Augmentez le trafic sur votre blog !

Google Blog Search (http://blogsearch.google.fr/) est un moteur de recherche consacré aux blogs, l'un des nombreux services proposés par la célèbre firme californienne....

Voir l'article...

- **GNU/Linux Magazine**

  - GNU/Linux Magazine N°113 - Février 2009 - Chez votre marchand de journaux
  - Édito : GNU/Linux Magazine 113
  - Un petit sondage pour améliorer nos magazines
  - GNU/Linux Magazine HS N°40 - Janvier/Février 2009 - Chez votre marchand de journaux
  - Edito : GNU/Linux Magazine HS 40

- **GNU/Linux Pratique**

  - Linux Pratique HS N°17 - Mars/Avril 2009 - Chez votre marchand de journaux
  - Édito : Linux Pratique HS N°17
  - Linux Pratique HS 17 - Communiqué de presse
  - Linux Pratique Essentiel N°6 - Février/Mars 2009 - Chez votre marchand de journaux
  - Édito : Linux Pratique Essentiel N°6

- **MISC Magazine**

  - Un petit sondage pour améliorer nos magazines
  - MISC N°41 : La cybercriminalité ...ou quand le net se met au crime organisé - Janvier/Février 2009 - Chez votre marchand de journaux

- Édito : Misc 41
- MISC 41 - Communiqué de presse
- Les Éditions Diamond adhèrent à l'APRIL !