

# **Contrôle d'accès e le POSIX Access Control Lists(ACL)**

CS435 - Administration de Système

Dan Pham et Fabrício Nascimento

Octobre 2009

## Introduction

Quand on désire contrôler l'accès aux données dans un système de fichiers, il y a plusieurs moyens d'y parvenir. Par défaut, les systèmes POSIX (Portable Operation System Interface)[2, 3] ont un mécanisme qui permet d'associer chaque entité avec un ensemble de règles, lequel est composé par une séquence d'octet qui exprime les droits du propriétaire, de son groupe et des autres utilisateurs.

Ce mode, traditionnel, assez simple est capable de résoudre les problèmes les plus fréquents. Par contre, il pose des limitations aux administrateurs de systèmes qui pour exprimer leurs besoins doivent employer des configurations non évidentes. Certaines applications choisissent de développer leur propre système de droit comme le serveur FTP Proftp[4] pour résoudre ce problèmes de droits.

Pour remédier à ces limitations, les systèmes UNIX peuvent employer les ACL. Cet article présente une exposition sur les ACL POSIX, ses modes de fonctionnement, ses qualités et désavantages. Le texte s'inspire de l'article d'Andreas Gruenbacher[1] qui a fait partit de l'équipe ayant ajouté le support aux ACL dans le noyaux Linux pour les systèmes de fichiers ext2 et ext3, qui sont les plus utilisés dans les monde UNIX.

## 1 Le POSIX 1003.1

Traditionnellement les systèmes qui implémentaient la norme POSIX avaient un système simple et puissant de permissions mais qui cependant posait certains problèmes. En effet, les différentes versions d'ACL disponibles étaient incompatibles entre elles.

Pour normaliser les problème de sécurité sur les systèmes POSIX (ACL en faisant partie), un groupe a été formé pendant la définition de la famille de normes POSIX 1003.1. Les premiers documents POSIX qui ont pris en compte ces questions étaient les documents 1003.1e (*System Application Programming Interface*) et 1003.2c (*Shell and Utilities*), cependant, le premier draft était trop ambitieux. En effet, le groupe responsable pour la normalisation avait divisé ses efforts sur un grand nombre de domaines qui comportaient les *Access Control Lists* (ACL), les *Audit*, les *Capability*, les *Mandatory Access Control* (MAC), et l'*Information Labeling*[1].

En Janvier de 1998[1] le financement pour ce projet à été suspendu, par contre, le travail n'était pas prêt. De toute façon le dixseptieme draft a quand même été rendu public[5].

Après cette publication, des systèmes UNIX appelés "*trusted*" (Trusted Solaris, Trusted Irix, Trusted AIX) ont été développés à partir du draft 17. Ces systèmes ne sont pas complètement compatibles entre eux. Heureusement aujourd'hui la plupart des systèmes UNIX et UNIX-like supportent les ACL. Ces implémentations sont usuellement compatibles avec le draft 17. Le projet TrustedBSD implémente aussi les ACL sur les système BSD. Les ACL sont apparues sur les Macs en 2003 avec la RELEASE MAC FreeBSD.

Les ACL sont une évolution du système de permissions traditionnel présent dans pratiquement tous les systèmes UNIX, alors, avant d'expliquer les ACL on va d'abord parler du modèle traditionnel.

### Système de permissions traditionnel

Le modèle traditionnel POSIX offre trois classes d'utilisateurs qui sont : le propriétaire (*owner*), le groupe propriétaire (*group*) et les autres utilisateurs (*others*). Chaque groupe a un octet que indique les permissions de lecture (*read*), d'écriture (*write*) et d'exécution (*execute*).

Après les trois octets peut venir le *Set User Id*, *Set Group Id* et le *Sticky Bit* qui peuvent être utilisés dans certain cas. Il faut faire attention avec le *Sticky Bit*, il permet aux utilisateurs normaux d'exécuter les utilitaires comme l'administrateur(*root*), donc une faille de sécurité dans une application utilisant le *Sticky Bit* peut compromettre le système entier.

Seul le *root* peut créer les groupes et changer les associations de groupes. Il peut aussi changer les propriétaires.

## Les ACL

Chaque ACL est une ensemble de règles d'accès. Dans une modèle de sécurité utilisant les ACL, si une entité fait une requête pour accéder aux données, il faut consulter la liste d'ACL pour savoir si nous avons la permission pour l'opération demandé. Les règles possibles peuvent être consultées dans le tableau ci-dessous(??).

Les types de ACL	
Type d'entrée	format
Propriétaire	user : :rwx
Utilisateur nommée	user :name :rwx
Groupe propriétaire	group : :rwx
Groupe nommée	group :name :rwx
Masque	mask : :rwx
Autres	other : :rwx

Les règles sont formées par un indicateur de classe (comme les classes du système traditionnel), l'identificateur pour préciser de quel utilisateur ou groupe on parle puis les octets de permissions.

Les ACL équivalentes au mode simple de permissions s'appellent les ACL minimales. Une ACL minimale possède 3 entrée (propriétaire, groupe propriétaire et autres) qui sont trivialement convertible. Si les ACL possèdent des entrées supplémentaires, on les appelle ACL étendues. Toutes les ACL étendues doivent avoir une entrée masque et peuvent contenir théoriquement autant d'entrées que l'on désire. On verra après que ce numéro d'entrée peut-être limitée pour chaque implémentation et qu'il est aussi important pour les performances.

Quelquefois on a des application qui ne sont pas conscient de les ACL, ça veut dire qu'on doit crée une relation entre les ACL e le système traditionnelle de façon que ces application n'irons commencer a donner tout a coup et de façon inattendue plusieurs droits aux utilisateurs ou groupes.

Quand on parle des ACL minimales ça n'arrive jamais, alors que la relation entre ces ACL e le système traditionnelle é directe. Par contre, dans les ACL étendu ça pose une problème. Pour le solutionner, on va changer le sens da classe groupe. Les propriétaire et les autres seront directement associe avec les classes de même nom, par contre, les entré de groupe et utilisateur nommé avec le groupe propriétaire seront tous associe avec la classe du groupe propriétaire. Par conséquent le sens du classe groupe propriétaire doit être redéfini comme le limite supérieur de les permission de chaque entrée dans les entrée associe a classe du groupe.

Comme dans les ACL étendues on peut avoir des entrées avec plusieurs utilisateurs et/ou groupes nomme, quelques de cette entrées peut-être contenir permissions qui la classe groupe n'aurais pas, alors, on peut avoir une inconsistance dans certain cas.

Ce question serais résoudre avec l'utilisation de une masque. Comme on peut l'observer dans le figure (1), il y a deux cas : Les ACL minimales où la classe groupe est référencée pour l'entrée du groupe propriétaire. Les ACL étendues

de la classe groupe seront trouvées en faisant un masque avec les permissions du groupe propriétaire et les permissions des utilisateurs et groupe nommés. Cela rend difficile le calcul de classe groupe.

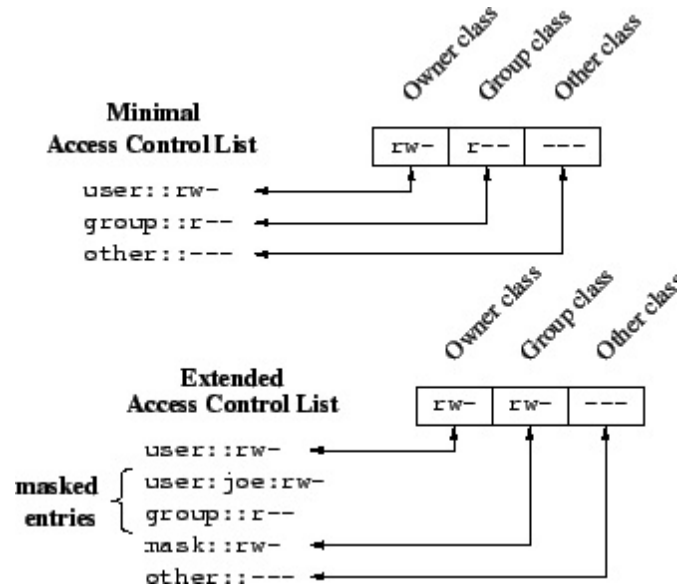


FIG. 1 – caption

Pour assurer la cohérence, quand une application change les permissions (par exemple le commande *chmod*) les ACL sont modifiée de façon a reproduire cette modification.

On a dit que la permission de la classe groupe est calculée comme la limite supérieure de tous les entrées dans le classe groupe. Avec les ACLs minimaux cette computation est simple, par contre, avec les ACLs étendu, on a besoin de masquer les permissions. Comme l'exemple de le tableau (??), les entrées de permission qui sont partie de la classe de groupe et qui aussi sont présente dans l'entrée masque sont applique effectivement. Si une permission était absent dans le masque, c'est a dire que aucun entrée de groupe (qui non le groupe du propriétaire) peut avoir ce permission, on dit dans ce cas qui la entrée est masquée.

La masque de permissionL		
Type	Format	Permission
Utilisateur nommée	user :jean :r-x	r-x
Masque	mask : :rw-	rw-
Permission Effective		r-

## Algorithme de vérification

Pour vérifier les droits d'accès d'un objet du système de fichier, il y a un algorithme assez simple.

---

**Algorithm 1** Vérifie si un utilisateur peut ou ne peut pas accéder à un objet du système de fichier

---

```
if l'identifiant de l'utilisateur du processus est le propriétaire then
    l'entrée du propriétaire détermine l'accès
else if l'identifiant d'utilisateur du processus correspond à une entrée d'utilisateur nommé dans la table des ACL then
    l'entrée détermine l'accès
else if un des identifiants de groupe du processus correspond au groupe propriétaire et l'entrée contient les permissions requises then
    l'entrée détermine l'accès
else if un des identifiants de groupes correspond à un des groupes nommés et cette entrée contient les permissions requises then
    l'entrée détermine l'accès
else if Un des identifiants de groupe du processus correspond au groupe propriétaire ou correspond à un des groupes nommés mais ni le groupe propriétaire ni aucun des groupes nommé contient les permissions requises. then
    ceci détermine que l'accès est interdit
else
    l'entrée other détermine l'accès
end if
if l'entrée qui détermine l'accès est l'entrée du propriétaire ou l'entrée other qui contient les permissions requises then
    l'accès est autorisé
else if l'entrée correspondante est l'utilisateur nom, ou le groupe propriétaire ou le groupe nommé et cette entrée contient les permissions requises et l'entrée masque contient aussi les permissions. (ou il n'y a pas d'entrée masque) then
    l'accès est autorisé
else
    l'accès n'est pas autorisé
end if
```

---

## Héritage mécanisme

Le système POSIX règle non seulement les droits d'accès aux objets du système de fichiers, mais aussi le mécanisme d'héritage. Les ACL sont partagés en deux types, les *access ACL* (qu'on a vu jusqu'à maintenant) et les *default ACL* qui comprennent les règles d'héritage.

Quand on parle de l'héritage, on parle des droits qui sont attribués aux objets du système de fichiers au moment où ils sont créés. Il y a un seul type d'objet qui peut être associé avec les *default ACL*; les répertoires. Il faut dire

que il n'y a pas de sens pour les *default ACL* pour les fichiers car on ne peut pas créer un fichier à l'intérieur d'un fichier. Aussi les *Default ACL* et les *access ACL* sont complètement indépendant.

Si un répertoire est crée dans une autre, si le première répertoire a *default ACL*, avec le mécanisme d'héritage, le deuxième aura le même ACL que le premier (*default* et *access*). Les objets qui ne sont pas des répertoires, devons hériter les *default ACL* seulement.

Chaque *system call* qui crée les objets du système de fichier a un *mode parameter*. Ce paramètre peut contenir neuf octets de permission pour chaque classe (propriétaire, groupe et les autres). Les permissions de chaque objet créée sont l'intersection des permissions définies pour les *default ACL* et le *mode parameter*.

Le système traditionnel a une commande pour désigner les modes de permissions par défaut pour les nouveaux fichiers et répertoires : le commande *umask*. Quand il n'y a aucune *default ACL*, la permission effective est déterminé par le *mode parameter* moins les permissions configurés avec *umask*.

## 2 ACL en use

Dans cette session on verrais les uses des ACL dans les système d'aujourd'hui.

### 2.1 ACL Kernel Patches

Les ACL *patches* ont été ajouter dans le noyaux Linux depuis November 2002. Cette *patches* implémentent le POSIX 1003.1e brouillon 17 et elles ont été ajoute dans le version 2.5.46 du noyaux. Donc le support ACL et aussi présent dans le dernière version du noyaux aujourd'hui. Depuis 2004 le support aux ACL étions disponible pour les système de fichier Ext2, Ext3, IBM JFS, ReiserFS et SGI XFS. Les ACL sont supporte aussi pour le système NFS, par contre, il y a quelques problèmes de sécurité connu[8].

Aujourd'hui c'est assez simple pour ajouter le supporte aux ACL dans les distribution Linux comme Ubuntu ou Debian. On verrais les pas pour ajouter ce supporte après.

### 2.2 Mac OSX

Le système de exploitation Mac OSX (10.6.2 Snow Leopard dans le moment de écriture de ce article) a aussi les supporte aux ACL complètement intégrée dans l'interface de utilisateur (2).

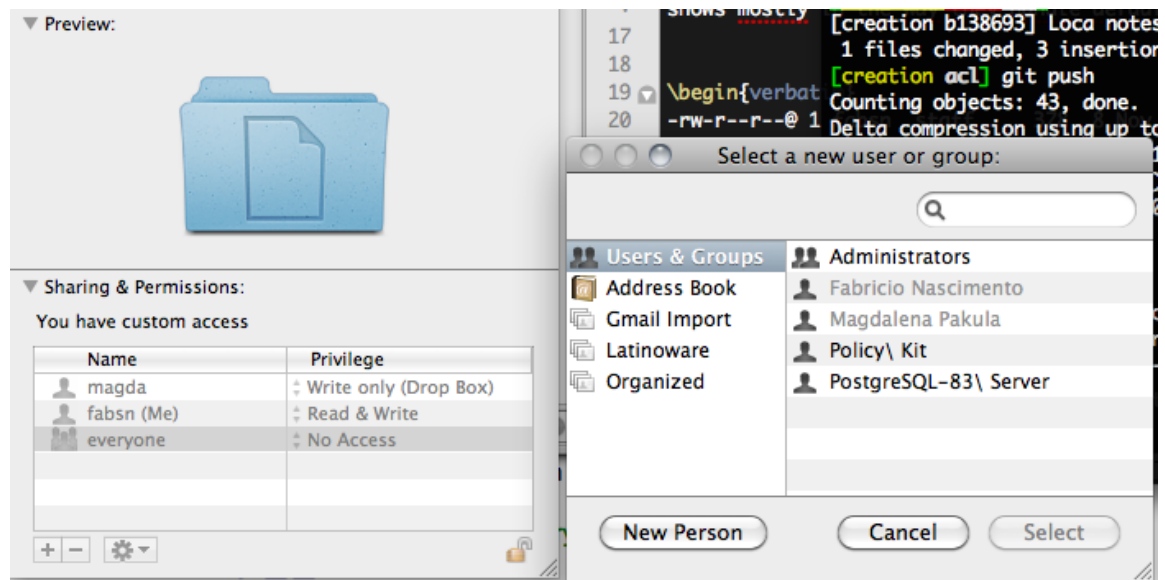


FIG. 2 – Mac OSX Snow Leopard ACL Interface



## Using ACL in Linux

Les dernière version des distribution Debian ou Ubuntu, comme Ubuntu 9.10, déjà vient avec le supporte aux ACL. Dans le Ubuntu 8.10 l'application Nautilus, qui est responsable pour la visualisation du système de fichier, contenait une interface pour les ACL, apparemment l'interface a été discontinuée et le Nautilus du Ubuntu 9.10 n'en y a pas encore. Les pas pour ajouter le supporte dans le Ubuntu 9.10 sont :

1) Installer le paquet des acl.

```
user@ubuntu:$ sudo apt-get install acl
```

2) Ajouter le option 'acl' au système de fichier correcte dans le /etc/fstab, comment: `UUID='gros sequence' /dev/hda6 /home ext3 rw,auto,acl 0 1`

3) Remonter le système de fichier avec le nouvelle option

```
user@ubuntu:$ sudo mount /home -o remount
```

## Ajouter ACL aux fichiers

On peut utiliser le commande 'ls -la' pour regarde les permission. Si une fichier contient information de sécurité avancée (comme *access list*) on va voir le "character" '+', comment dans le sortie du command 'ls' ci-dessous (2.2). Une fichier avec '@' était dire que le fichier a quelque EAs.

```
-rw-r--r--@ 1 fabsn staff    378  8 Nov 15:29 Makefile
-rw-r--r--@ 1 fabsn staff    618  8 Nov 15:59 README
-rw-r--r--@ 1 fabsn staff     31  8 Nov 15:15 draft-header
-rw-r--r--@ 1 fabsn staff     24  8 Nov 15:15 header
drwxr-xr-x 2 fabsn staff    102  8 Nov 15:26 img
-rw-r--r-- 1 fabsn staff    972  8 Nov 15:57 rapport-draft.aux
-rw-r--r-- 1 fabsn staff  18129  8 Nov 15:57 rapport-draft.log
drwxrwxr-x+ 3 fabsn staff   1024  8 Nov 20:23 repertoire
```

Pour voir les ACL on doit utilise le commande *getfacl*. Regarde que les information sont ajoute d'accord avec les définition dans l'introduction sur les ACL dans la table 1.

```
fabsn@vadmin:/media/esisar$ getfacl repertoire/
# file: repertoire/
# owner: root
# group: root
user::r-x
user:daemon:rwX
user:bin:rwX
user:fabsn:rwX
```

```
user:nobody:rwX
group::r-x
group:admin:rwX
group:fabsn:rwX
mask::rwX
other::r-x
```

Aussi on a la commande *setfacl* pour modifier, ou ajouter les permission ACL. Le commande dessous par exemple modifie (-m) les permission du utilisateur *fabsn* pour le répertoire.

```
setfacl -m u:fabsn:r-x repertoire
```

## Exemple

Voici une exemple d'utilisation trouvable dans le article d'Andreas Gruembacher[1].

On parte de la création de un répertoire avec l'application de umask de valeur 027 (octal). Ça veut dire que il va désactiver l'écriture pour le groupe propriété et l'écriture, la lecture e l'exécution pour les autres.

```
$ umask 027
$ mkdir dir
$ ls -dl repertoire
drwxr-x--- ... user groupe ... repertoire
```

La lettre "d" montre que on l'objet "répertoire" c'est une répertoire suivi pour les octet de permission "rwxr-x---". Les réticences dans le command supprime les information avec aucune pertinence pour cette exemple. Cette permission basic on toujours une représentation dans les ACL, pour les afficher on faire :

```
$ getfacl repertoire
# file: repertoire
# owner: user
# group: groupe
user::rwX
group::r-x
other::---
```

Le trois première ligne après le commande (démarrer pour le #) nous donne les information sur les propriétaire (groupe e utilisateur) et le nom du fichier. Après chaque ligne vient la liste des ACL. Ce exemple la ce de une ACL

### 3 Implémentations

Les ACLs sont fréquemment implémentées comme extensions du noyau, c'est à dire des modules un système LINUX. L'objectif de cette section est d'expliquer de manière globale l'implémentations des ACL.

—————Le discussion doit lancer la base pour les évaluations de performance et les problèmes dans les sessions prochaines.—————

"Les ACLs sont des informations de taille variable qui sont associées avec les objets du système de fichier"[1]. Plusieurs implémentations des ACLs sont possibles. Par exemple, avec Solaris, dans le système de fichier UFS[6] chaque *inode* peut avoir une ACL. S'il en a une, il doit avoir l'information *i\_shadow*, un pointeur pour un *shadow inode*. Les *shadow inode* sont comment fichiers réguliers d'utilisateurs. Différent fichiers avec les mêmes ACL peut avoir pointeurs pour le même *shadow inodes*. Les information des ACL sont garde dans les bloc de données de chaque *shadow inodes*.

La capacité d'associer des informations avec des fichiers est utilisé dans plusieurs fonctions du système de exploitation. De ce fait, la plupart des systèmes *UNIX-like* (de type Linux) on trouve les Attributs ... tendus (*Extended Attributes (EAs)*). Les ACL sont implémentées avec ce mécanisme.

La manpage [1] *attr(5)* contient des explications précises sur les EAs dans Linux, ———au notre but, suffit dire que comme les variables des processus, les EAs sont des paires (nom, valeur) associées de manière persistantes avec les objets du système de fichiers et que les appels Linux, dans le espace de utilisateur, sont employé pour opérer sur les information de ces paires dans le espace de adresse du noyaux. Aussi pour l'implémentation de cette infrastructure dans les système FreeBSD il faut voir le article de Robert Watson[7]. ———

—————Cet article contient aussi une comparaison de plusieurs implémentations de ces systèmes.

Dans le monde linux, ajouter le support aux ACL avec une version limitée des EA offre plusieurs avantages : un grande facilité d'implémentation, ———opération atomique et interface *stateless* que laisse aucun surcharge à cause de les *file handlers*. On verra après dans la section de performance, que l'efficacité est assez importante pour être oublier quand on parle de les données fréquemment accés comme les ACL. ———

#### 3.1 Les EAs et les systèmes de fichiers

Dans le monde UNIX, chaque système de fichier a une différent implémentation pour les EAs. On peut penser qu'une solution partagée pour l'ensemble des systèmes pourrait être plus efficace. Par exemple, si on prend une solution simple où chaque objet du système de fichiers a les EAs, un répertoire avec un fichier qui a le clés EA comment le nom et le contenu comment le valeur. Cette implémentation consommerait beaucoup de espace, étant donné que les blocks du système de fichier seront gaspillés pour conserver petit morceaux de données, aussi ce solution perdrait les temps pour chercher ces informations à chaque accés

de fichier. Aux frais de ces problèmes chaque système tire profit de ces qualités pour ajouter le supporte aux EAs.

### **Ext (2,3 et 4)**

Les ACL dans Ext suivent le principe linux : "La solution la plus simple qui marche" et pour cette raison subviennent quelques limitations. D'autres solutions existent, par contre, elle sont difficiles à ajouter au noyau de manière satisfaisante[9].

La solution actuelle ajoute aux *i\_node* une entrée qui s'appelle *i\_file\_acl*. Cette entrée, si différent de 0, est une ponteur sur un bloc d'EAs. Ce bloc d'EAs a les informations de nom et valeur de tous les ACL du fichier indiquè pour cet *i\_node*.

Le mécanisme a aussi une optimisation. Deux fichiers avec le même ensemble de ACL point vers le même bloc d'EAs. Le système garde une *hash map* avec les *checksum* des blocs d'EAs et leurs adresse. Chaque block a aussi un compteur de référence, comme les liens *hard*. Ce mécanisme détermine aussi que ce compteur là ne peut pas avoir plus que 1024 références. Il s'agit d'une mesure de sécurité en cas de perte des données.

Aussi une limitation est imposée : toutes les données des EAs d'un fichier doivent occuper un bloc d'EAs ayant une taille de 1, 2 ou 4 KBs.

### **JFS**

Dans JFS, les EAs sont ajoutées dans une liste consécutive de blocs contigus (un extent). Cela veut dire que chaque paire (nom,valeur) est gardée en séquence et que chaque valeur de la paire ne peut pas être plus grande que 64kb. Si les EAs sont assez petites, elles pourront être gardées dans le même lieu que les informations du fichier. De ce façon, il n'y a pas les limitations d'ext3.

### **XFS**

XFS est sans aucun doute le système de fichiers le plus simple pour implémenter les EA. Les paires d'EA de petite tailles sont stockées directement dans l'inode, celles de taille moyenne sont stockées dans les blocs feuilles de l'arbre binaire et pour celle de grande taille, dans un arbre binaire complet. —manque—  
— XFS peut configurer la taille de sa table d'inodes. La taille minimale est de 256 octet et la taille maximale peut aller jusqu'à la moitié des blocs du système de fichier. Dans le cas où on a une table de taille minimale, celle-ci n'est pas assez grande pour accueillir les ACLs. On doit alors les stocker de manière externe —au système de fichier—. Si on augmente la taille de la table les ACL pourront y être stockées. Les ACL étant très souvent interrogées par le système, cela augmente les performances en terme de temps d'accès au détriment de l'espace disque qu'elles consomment. XFS n'a pas de mécanisme de partage des attributs. La taille individuelle des attributs est limitée à 64Kb.

## ReiserFS

ReiserFS support le *tail merging* qui permet à plusieurs fichiers de partager le même bloc pour stocker leurs données. Cela rend le système très efficace pour s'il on possède de nombreux fichiers de petite taille. De l'autre côté cette technique consomme beaucoup de ressources CPU.

Comme le ReiserFS peut facilement manipuler des petits fichiers, les EA peuvent être implémentées sous forme de petits fichiers. Pour chaque fichier qui a un EA, un dossier spécial (qui est souvent caché) est créé avec un nom dérivé de son inode. Dans ce dossier chaque EA est stockée dans un fichier séparé qui a pour nom le nom de l'attribut. Le contenu de chaque fichier est la valeur de l'attribut.

Le système ReiserFS n'implémente pas le partage d'attributs mais la création d'une extension pour le gérer est possible. La taille individuelle des attributs est limitée à 64Kb.

—————-Sharing could even be implemented on a per-attribute bases, so the result would be a highly efficient and flexible solution. —————The size of individual attributes is limited to 64 KiB.

## HGFS+

## Samba

## Conclusion

## Références

- [1] Andreas Gruenbacher, *POSIX Access Control Lists on Linux*. <http://www.suse.de/~agruen/acl/linux-acls/online/>, 2003.
- [2] IEEE Std 1003.1-2001 (Open Group Technical Standard, Issue 6), Standard for Information Technology–Portable Operating System Interface (POSIX) 2001. ISBN 0-7381-3010-9. <http://www.ieee.org/>
- [3] IEEE 1003.1e and 1003.2c : Draft Standard for Information Technology–Portable Operating System Interface (POSIX)–Part 1 : System Application Program Interface (API) and Part 2 : Shell and Utilities, draft 17 (withdrawn). October 1997. <http://wt.xpilot.org/publications/posix.1e/>
- [4] Mark Lowes : Proftpd : A User's Guide March 31, 2003. <http://proftpd.linux.co.uk/>
- [5] Winfried Trümper : Summary about Posix.1e. Publicly available copies of POSIX 1003.1e/1003.2c. February 28, 1999. <http://wt.xpilot.org/publications/posix.1e/>
- [6] Jim Mauro : Controlling permissions with ACLs. Describes internals of UFS's shadow inode concept. SunWorld Online, June 1998.
- [7] Robert N. M. Watson : Introducing Supporting Infrastructure for Trusted Operating System Support in FreeBSD. BSDCon 2000, Monterey, CA, September 8, 2000. <http://www.trustedbsd.org/docs.html>
- [8] Andreas Grünbacher : Linux Extended Attributes and ACLs. Session "Known Problems and Bugs". <http://acl.bestbits.at/problems.html>
- [9] Andreas Dilger : [RFC] new design for EA on-disk format. Mailing list communication, July 10, 2002. <http://acl.bestbits.at/pipermail/acl-devel/2002-July/001077.html>
- [10] Austin Common Standards Revision Group. <http://www.opengroup.org/austin/>