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 Gruembacher[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 dixsèptieme 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 (***r**ead*), d'écriture (***w**rite*) et d'exécution (*e**x**ecute*).

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 les 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) qu'on peut convertir directement. Si les ACL possèdent des entrées supplémentaires, ont les appelle ACL étendues. Toutes les ACL étendues doivent avoir une entrée masque et peuvent contenir théoriquement autant d'entrées que l'on désire. 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 programmer pour fonctionner avec les ACL, cela veut dire que l'on doit créer une relation pour convertir

les ACL vers le système traditionnel, de façon que ces application ne donnerons pas de façon inattendue plusieurs droits aux utilisateurs ou groupes.

Quand le programme utilise des ACL minimales ce problème n'arrive jamais : la relation entre ces ACL et le système traditionnel est directe. Par contre, dans les ACL étendues cela pose une problème. Les propriétaires et les autres seront directement associe avec les classes de même nom, par contre, les entrées de groupe et utilisateur nommé avec le groupe propriétaire seront tous associés avec la classe du groupe propriétaire. Par conséquent, le sens du classe groupe propriétaire doit être redéfinie comme la limite supérieure des permissions de chaque entrée dans pour la classe groupe.

Comme dans les ACL étendues on peut avoir des entrées avec plusieurs utilisateurs et/ou groupes nommés, quelques unes ces entrées peuvent contenir des permissions que la classe groupe n'aura pas, alors, on peut avoir des incohérences dans certain cas.

Cette question peut se résoudre avec l'utilisation d'un 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, les permissions des utilisateurs nommés et groupe nommés. Cela rend difficil le calcul de classe groupe.

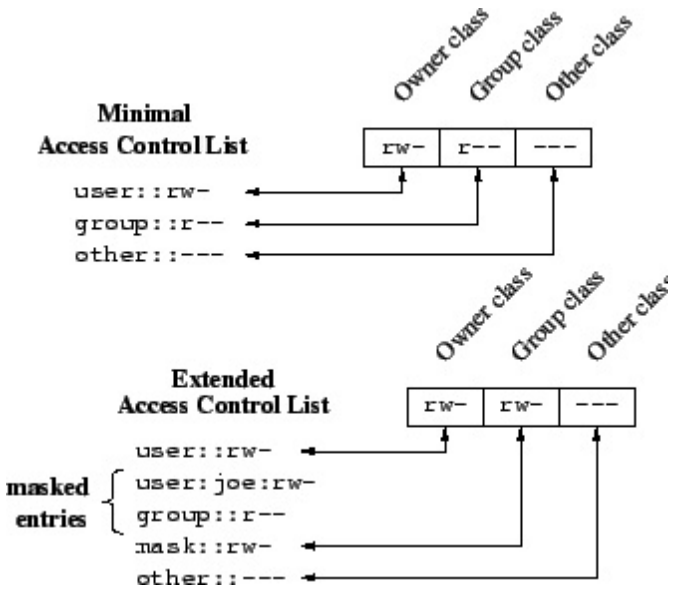

FIG. 1 – caption

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

On a dit que les permissions du masque sont calculées comme la limite supérieure de tous les entrées dans le classe groupe. Avec les ACLs étendus, on a besoin de masquer les permissions. Comme l'exemple du tableau (**??**), les permission des entrées de la classe groupe et qui aussi sont présentes dans le masque sont appliquées. Si une permission est absente dans le masque, c'est à dire qu'aucune entrée de groupe ne peut avoir cette permission, on dit dans ce cas que l'entrée est masquée.

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

Par être consistent, dans une application que utilise les ACL étendu, le permission du groupe propriétaire est toujours calculée comme la union entre le permission de ce groupe e la masque.

Algorithme de vérification

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

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 ACL d'accès (qu'on a vu jusqu'à maintenant) et les ACL par défaut qui comprennent les règles d'héritage.

Quand on parle de l'héritage, on parle des droits qui sont attribués aux objets du systèmes de fichiers au moment où ils sont crées. Il y a un seul type d'objet qui peut être associe avec les ACL par défaut les répertoires. Il faut dire que il n'y a pas de sens pour les ACL par défaut pour les fichiers car on ne peut pas créer un fichier à l'intérieur d'un fichier. Aussi les ACL par défaut et les ACL d'accès sont complètement indépendant.

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

Algorithm 1 Vérifie se une utilisateur peut ou ne peut pas accéder une 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 groupe 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 permission requises. **then**
  ceci détermine que l'accès est interdit
else
  l'entrée autre détermine l'accès
end if
if l'entrée qui détermine l'accès est l'entre du propriétaire ou l'entrée autres 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 permission. (ou il n'y a pas d'entrée masque) **then**
  l'accès est autorisé
else
  l'accès n'est pas autorisé
end if

Quand il n'y a aucune ACL par défaut, la permission effective est déterminé par le *mode parameter* moins les permissions configurés avec *umask*.

2 Implèmentassions

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

"Les ACLs sont des informations de taille variable qui sont associées avec les objets du systéme 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'l 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 utilise dans plusieurs fonctions du système de exploitation. De ce fait, la plupart des systèmes *UNIX-like* (de type Linux) on trouve les Attributs étendus (*Extended Attributes (EAs)*). Les ACL sont implémentées avec ce mécanisme.

La manpage [1] *attr(5)* contient des explications précises sur les EAs dans Linux. Comme les variables des processus, les EAs sont des paires (nom, valeur) associées de manière persistantes avec les objets du système de fichiers. Les appels système Linux pour les ACL, sont employées pour opérer sur les informations contenues dans ces paires dans le espace d'adresse du noyaux. Aussi l'implémentation des EA et des appels système dans FreeBSD sont documentés par l'article de Robert Watson[7]. Cette article content aussi une comparaison de plusieurs implèmentations de ces système.

Dans le monde linux, ajouter le support aux ACL en les implémentant avec des EA limitée offre plusieurs avantages : un grande facilité d'implémentation, les opération sont atomiques et l'interface est sans état donc il n'y a aucune surcharge au niveau *file handlers*. On ne doit pas oublier que l'implémentation doit être efficace car les ACL souvent utilisées.

2.1 Les EAs et les systémes 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 donne que les blocks du système de fichier seront gaspilles pour conserver petit morceaux de donne, aussi ce solution perdrait les temps pour chercher ces information a

chaque accès 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 ponter sur un bloc d'EAs. Ce bloc d'EAs a les informations de nom et valeur de tous les ACL du fichier indique 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 entent). 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 pourrons 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.

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 dans l'arbre binaire. 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 côte 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 fichiers qui a un EA, un dossier spécial (qui est souvent cache) est créé avec un nom dérive de son inode. Dans ce dossier chaque EA est stockée dans un fichier sépare 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.

Système de fichiers à distance

Samba

NFS

3 ACL en use

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

3.1 ACL Kernel Patches

Les ACL *patchs* ont été ajouté dans les noyaux Linux depuis Novembre 2002. Ces *patchs* implémentent le POSIX 1003.1e draft 17 et elles ont été ajoutées dans la version 2.5.46 du noyau. Donc le support des ACL et aussi présent dans le dernière version du noyau actuel. Depuis 2004 le support aux ACL est disponible pour les système de fichiers Ext2, Ext3, IBM JFS, ReiserFS et SGI XFS. Les ACL sont supportés aussi pour le système NFS, par contre, quelques problèmes de sécurité sont connus[8].

Aujourd'hui il est assez simple d'ajouter le support aux ACL dans les distributions Linux comme Ubuntu ou Debian. On verra marche à suivre pour ajouter ce support après.

3.2 Mac OSX

Le système de exploitation Mac OSX (10.6.2 Snow Leopard au moment où ces lignes sont écrites) supporte aussi les ACL et elles sont complètement intégrées à l'interface de l'utilisateur (2).

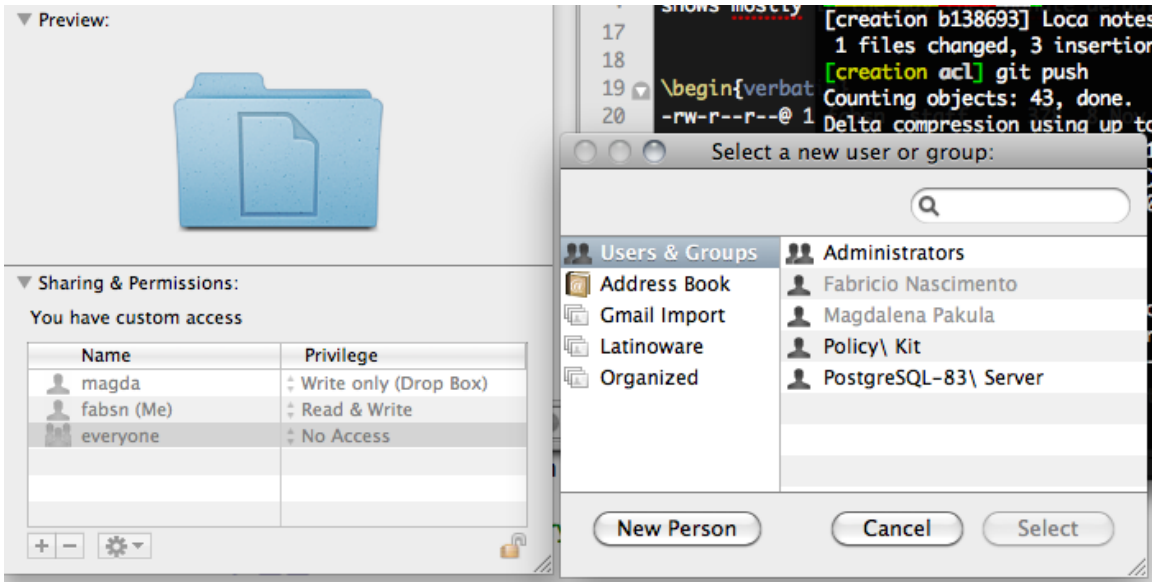

FIG. 2 – Mac OSX Snow Leopard ACL Interface

Utiliser les ACL sous Linux

Les dernières versions des distributions de Debian ou Ubuntu, comme Ubuntu 9.10, viennent avec le support des ACL. Dans Ubuntu 8.10 l'application Nautilus, qui est responsable pour la visualisation du système de fichiers, contenait une interface pour les ACL, qui a apparemment disparues dans les version suivantes et Nautilus dans Ubuntu 9.10 ne l'intègre pas. La marche à suivre pour ajouter le support des ACL dans le Ubuntu 9.10 est la suivante :

```
1) Installer le paquet des acl.
user@ubuntu:$ sudo apt-get install acl

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

3) Remonter le système de fichiers avec la nouvelle option
user@ubuntu:$ sudo mount /home -o remount
```

Ajouter les ACL aux fichiers

On peut utiliser le commande 'ls -la' pour regarde les permissions. Si un fichier contient information de sécurité avancée (comme les ACL) on va voir le caractère '+', comme dans la sortie du command 'ls' ci-dessous (3.2). Un fichier avec devant une '@' veut dire que le fichier contient 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 utiliser la commande *getfacl*. Regarde que les information sont ajoute d'accord avec les définition dans l'introduction sur les ACL dans la tabelle 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 le 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 ACL d'accès

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 group ... 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: goup
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 montre une ACL minimale. Si par exemple on ajoute permissions pour le utilisateur "jean" avec le commande *setfacl* on va avoir les ACL étendu.

```
$ setfacl -m user:jean:rwx repertoire
$ getfacl --omit-header repertoire
user::rwx
user:jean:rwx
group::r-x
mask::rwx
other::---
```

Il faut rappeler que le command *setfacl* a été employer avec le modificateur *-m (modify)*. Pour regarder les résultat on utilisé une autre fois le *getfacl*. Il faut savoir que le modificateur *–omit-header* cache les première 3 ligne avec les information sur les propriétaires.

Tout d'abord on peut voir que l'entrée masque a été ajouter avec l'entrée d'utilisateur jean. Ces permission sont crée comme la union entré les permission de jean et du groupe propriétaire. La masque doit être une valeur que ne masque aucune permission, alors, ce valeur se trouve comme l'union de les élément que définissions la classe groupe.

```
ls -dl repertoire
drwxrwx---+ ... user group ... repertoire
```

Si on rappel le dernière exemple, le groupe propriétaire n'y a pas le droit de écriture (*group : :r-x*), par contre, les groupe classe vient avec ce droit. C'est pour ça que quand on calcule effectivement le droit pour le groupe propriétaire dans le modèle acl, ce droit est toujours le union avec la masque.

Le prochaine exemple montre comme lest ACL sont modifiée pour l'emploie du command *chmod* ou *setfacl*. On va effacer le permission de écrit de la classe groupe. On doit voir que si il n'y a pas de masque, le command doit changer directement les permission dé l'entrée ACL du groupe propriétaire.

```
$ chmod g-w repertoire
$ ls -dl repertoire
drwxr-x---+ ... user group ... repertoire
$ getfacl --omit-header repertoire
user::rwx
user:jean:rwx  #effective:r-x
group::r-x
mask::r-x
other::---
```

Si une ACL entrée contient permission désactivé pour le masque, getfacl doit ajouter une commentaire qui montre ça différence. Il faut voir aussi qu'est-ce que se passe quand on rajoute le permission.

```
$ chmod g+w repertoire
$ ls -dl repertoire
drwxrwx---+ ... user group ... repertoire
$ getfacl --omit-header repertoir
user::rwx
user:jean:rwx
group::r-x
mask::rwx
other::---
```

Cette exemple montre que les changes avec le commande *chmod* ne sont pas destructives. Les opérations sont complètement réversibles et les permission avant et après les deux opération d'aller et retour sont les mêmes, une caractéristique très important de les ACL POSIX.

Exemple des ACL par défaut

Avec le modifier -d, on peut ajouter les ACL par défaut d'un répertoire.

```
$ setfacl -d -m group:admin:r-x repertoire
$ getfacl --omit-header repertoire
user::rwx
user:jean:rwx
group::r-x
mask::rwx
other::---
default:user::rwx
default:group::r-x
default:group:admin:r-x
default:mask::r-x
default:other::---
```

Les ACL par défaut vient après les ACL d'accès et sont préfixe pour le chaîne de caractères "default : :". Le courent ce que quand on ajoute une nouvelle règle seulement dans les ACL d'accès comme on a faire pour le groupe *admin* aucune chose change avec les ACL par défaut. Par contre, il y a une extension sur linux que ajoute de manière automatique le règle dans les ACL par défaut.

Il faut voir que il n y a pas aucun entrée pour jean dans les défaut ACL, alors, jean n'aura pas accès dans le nouvelle objet crée dans le répertoire (sauf si il est membre de un groupe qui a les permission).

D'accord avec le mécanisme d'héritage qu'on a vu dans le page 1, le sub-repertoire doit hériter les ACL de sont parent. Par défaut, le command *mkdir* utilise une *mode parameter* de 0777 pour ce appel de système. Observez :

```
$ mkdir repertoire/subrep
$ getfacl --omit-header repertoire/subrep
user::rwx
group::r-x
group:admin:r-x
mask::r-x
other::---
default:user::rwx
default:group::r-x
default:group:toolies:r-x
default:mask::r-x
default:other::---
```

Pour les fichiers ç'arrive aussi. Le commande *touch* utilise le *mode value* 0666. Tous les permission que ne sont pas présent dans le *mode parameter* sont écraser.

Enfin, aucune permission a été écraser dans le groupe classe, cependant, la masque a été appliqué. Ce politique s'assure que par exemple les application comme les compilateurs peuvent marcher bien avec les ACL, alors que même que ils ne supportent ce mécanisme, il peuvent créer les fichier avec les permission restreint et après les donner les permission d'exécution et quand même, les groupes et autres permission doivent marcher comme attendu.

Problèmes

Compatibilité

Il y deux situation qui devons être considère quand on parle de j'ajutage les nouvelle fonctionnalité aux système de fichiers et consequentent au noyaux : les mises a jour et comme le système qui ne sont pas mise a jour répondre. Aussi il faut qu'on sache que quand on a besoin de un noyaux plus léger, par exemple pour démarré le système de récupération, et que dans cette cas on ne veut pas avoir les EAs dans le noyaux.

Tous les système de fichier que n'avons pas le supporte a les ACl activé on a mécanisme de mis en place le supporte dans le moment de montage or automatiquement dans le première use de les EAs, sans l'interférence d'utilisateur.

On espère que les vieux version du noyaux n'arrêtions pas de marcher avec les ACL. Dans un noyaux on les ACL ne sont pas supporté, les système de fichiers que implementent les ACL dans les répertoire , comme le ReiserFS, devient venir visible. Il faut quand même que les droit de permission de ces fichiers sont configure de façon que les utilisateurs ne peuvent pas changer les information de manière inadéquat. Un autre exemple c'est avec le système de fichier ext2 et ext3, si on effacer les fichier avec les ACL dans le kernel sans les support les information des ACL ne seront pas effacer, alors, on peut exécuter manuellement la vérification du système après pour mettre a jour les information.

Aussi le mécanisme de héritage ne marcherai pas.

Copie de Sécurité

Un aspect très important et usuellement oublier sont les copie de sécurité. Les outils comment *cpio* et *tar* ne connait pas les ACL. Ça veut dire que quand on doit garder notre information, si on en faire avec ces outils on perdra les EAs.

Une format appelle *pax (Portable Archive Interchange)* a été défini pour POSIX pour résoudre ce problème. Le outil *pax* peut comprendre les format de *cpio* et *tar* et aussi le nouveaux format *pax*. Ce nouveau format a les *extended headers* que peuvent décrire les EAs. Le outil *tar* peut le lire par contre il doit perdre les information de ACL.

On peau aussi utiliser *getfacl* et *setfacl* comment outils pour faire les copie de les droits d'accès, par contre, c'est pas pratique de les user si on a pas une copie de sécurité complète, mais quelques fichiers.

Références

[1] Andreas Gruenbacher, *POSIX Acess 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/