Initiation aux listes de contrôle d'accès (ACL) et aux attributs étendus (EA) sous Linux

Lionel Tricon - lionel.tricon@free.fr

Pendant longtemps, les choses n'ont pas beaucoup évoluées dans la gestion traditionnelle des droits d'accès sous Linux. Les raisons ? Le modèle proposé, solide et performant, qui a longtemps répondu à la majorité des besoins des utilisateurs.

Article publié dans GNU/Linux Magazine France, numéro 79 (Janvier 2006).

Préambule

Sous Linux (et sur toutes les plate-formes dérivées ou inspirées d'Unix), un fichier appartient toujours à un Propriétaire (*owner*) et se voit toujours associé à un Groupe (*owning group*) : Un groupe étant une notion abstraite permettant de regrouper un ensemble d'utilisateurs. A cela, il faut rajouter un troisième cas : Un utilisateur qui ne serait pas le propriétaire d'un fichier et qui n'appartiendrait pas au même groupe (de facto, tous les Autres utilisateurs : *other*). La règle veut que seul le propriétaire (en dehors du super-utilisateur, plus communément appelé « root ») soit autorisé à pouvoir assigner des droits de lecture, d'écriture ou d'exécution à un fichier dans chacunes des entités concernées (utilisateur, groupe et autres). On parle ici de fichier au sens large du terme, et donc incluant plus généralement les répertoires, liens et autres fichiers spéciaux.

Si vous exécutez dans une console la commande **ls -l** sur un fichier texte accessible en lecture et en écriture pour le propriétaire et les membres du groupe, et en lecture seule pour les autres utilisateurs, vous pourriez obtenir le résultat suivant :

```
-rw-rw-r-- 1 lionel users 12 2005-10-23 16:30 fichier.txt
```

Au premier coup d'oeil, on constate que le propriétaire du fichier est l'utilisateur **lionel** et que le fichier **fichier.txt** appartient au groupe **users** (on peut en déduire que l'utilisateur appartient au moins au groupe **users**).

La première colonne nous renseigne sur les droits appliqués au fichier : Cette colonne est constituée d'un caractère d'introduction permettant de préciser le type du fichier (ici, le caractère tiret est utilisé ce qui indique un fichier normal) suivi de trois blocs de trois caractères. Le premier bloc de trois caractères précise les droits du propriétaire du fichier (**classe utilisateur**), le second celui du groupe (**classe groupe**) et enfin le dernier s'adresse aux autres utilisateurs (**classe autres**).

Dans chacun de ces blocs, le premier caractère précise les **droits d'accès en lecture** du fichier (**r**), le second caractère les **droits d'écriture** (**w**) et enfin le dernier les **droits d'exécution** (**x**). Le caractère (**-**) est utilisé pour préciser que ces droits ne sont pas attribués (au passage, on constate que le droit d'exécution n'est assigné dans aucun des blocs dans notre exemple, ce qui est plutôt logique pour un simple fichier texte).

Nous prenons en exemple un fichier normal (**-**) mais nous aurions très bien pu faire référence à un **répertoire** (**d**), à un **lien** (**l**) ou encore à un **périphérique** de type **bloc** (**b**) ou de type **caractère** (**c**) sachant que ces deux derniers type de fichiers (appelés fichiers spéciaux) sont généralement cloisonnés dans le répertoire **/dev** (ces fichiers sont les points d'entrées qui nous permettent de communiquer avec le noyau et d'accèder aux interfaces du système).

Le cas d'un répertoire est un peu différent et la signification des droits va alors largement varier : Le droit de lecture (r) autorise cette fois l'entité à venir consulter en lecture le contenu du répertoire, le droit d'éxécution (x) l'autorise à entrer dans le répertoire (via la commande **cd** par exemple) et enfin le droit d'écriture (w) permet de créer de nouveaux fichiers dans ce répertoire.

Habituellement, sur un poste de travail, un répertoire donne les pleins pouvoirs au propriétaire (rwx) et permet un accès restreint en lecture seule au groupe du fichier et aux autres utilisateurs (r-x) comme l'illustre l'exemple suivant :

```
drwxr-xr-x 1 lionel users 35 Jui 21   15:43  Documents
```

Afin de traiter certains cas particuliers et étendre les droits standards **rwx**, trois attributs de fichier supplémentaires ont été rajoutés qui vont permettre en particulier d'usurper temporairement l'identité de l'utilisateur ou du groupe d'un fichier sous certaines circonstances.

Prenons le cas d'un binaire qu'il faut obligatoirement exécuter sous l'utilisateur root (l'exemple typique est le programme **ping** qui nécessite des permissions root pour pouvoir créer et injecter de nouveaux paquets ICMP dans la pile IP) : On va alors activer l'attribut de fichier **`set-uid`** sur le binaire pour indiquer au système qu'il doit être exécuté sous un utilisateur spécifique, en l'occurence celui du propriétaire du fichier (root dans notre cas). En pratique, l'utilisateur pourra alors lancer la commande **ping** sans être root (il ne

Préambulefaut bien évidemment pas utiliser cet attribut à tord et à travers : Cela rend votre système plus vulnérable en cas d'attaque).

Si les droits **`-rwsr-xr-x`** sont assignés au fichier **/bin/ping** appartenant à l'utilisateur root, cela signifie que le programme sera exécuté en tant qu'utilisateur root pour tous les utilisateurs.

Cette propriété est aussi disponible pour les groupes par l'intermédiaire de l'attribut **`set-gid`**. Un programme s'exécute alors sous les droits du groupe d'appartenance du fichier, cela quelque soit l'utilisateur qui l'a instancié. Cette fonctionnalité possède une propriété surprenante et très utile : Si on assigne l'attribut set-gid à un répertoire (par exemple, avec les droits **`drwxrwsr--`**), tous les fichiers et sous-répertoires créés à postériori seront attribués au groupe d'appartenance du répertoire.

Restons dans le domaine des répertoires en nous intéressant à l'attribut **`sticky`** qui se propose d'empêcher les utilisateurs d'effacer les fichiers dont ils ne sont pas propriétaires. Si cet attribut est assigné à un répertoire (par exemple, avec les droits **`drwxrwxrwt`**), les utilisateurs n'auront les droits d'effacer que les fichiers dont ils sont propriétaires (cet attribut est communément utilisé pour protéger les fichiers du répertoire **/tmp**).

Les attributs **`set-uid`** et **`set-gid`** sont activés par défaut lors du montage d'une partition (option **`suid`**) et il faut explicitement utiliser l'option **`nosuid`** dans le fichier **/etc/fstab** ou en ligne de commande dans l'appel à **mount** pour les rendre inopérants. Ce qui est notamment le cas lors du montage d'un CDROM ou d'une partition Windows, par exemple, pour des raisons évidentes de sécurité (tout comme on assigne par défaut un groupe et un utilisateur aux fichiers présents sur un support amovible ; sans cela, il serait trivial de prendre le contrôle d'une machine juste en branchant une clef USB).

Les droits d'accès à un fichier sont modifiables par appel de la commande **chmod** dans une console (comme nous l'avons vu précédemment, seul le propriétaire du fichier et le super-utilisateur root sont habilités à l'utiliser sur un fichier) ; des facilités graphiques ayant été prévues dans les navigateurs de fichiers des principaux bureaux Linux, en particulier KDE et GNOME. Par symétrie, les commandes permettant de régler les droits de propriétés des fichiers sont **chown** (*change owner*) et **chgrp** (*change group*).

La gestion des droits Unix, un modèle dépassé ?

Comme nous venons de le voir, la gestion des droits d'accès Unix (normé **POSIX**) est un mécanisme particulièrement efficace qui couvre l'essentiel des situations que l'on rencontre habituellement au quotidien (idéal dans une utilisation locale en poste de travail ou sur un serveur dédié) mais qui patit toutefois d'un manque criant de flexibilité dans un environnement multi-utilisateurs : Par exemple, lorsqu'il s'agit de permettre à un utilisateur, qui ne fait pas partie du groupe auquel appartient un fichier, d'obtenir certains droits temporaires sur le fichier (ce qui n'est pas fait pour faciliter le partage de fichiers au sein d'un réseau d'entreprise).

L'absence de gestion atomique des droits oblige souvent l'administrateur à rajouter l'utilisateur dans le groupe de la ressource à partager au risque de donner un blanc-seing à ce dernier qui ne va peut être pas se restreindre à la ressource en question.

On peut bien évidemment créer un groupe dédié pour partager cette ressource (ce qui paraît disproportionné !) ou éluder le problème en activant l'attribut **`set-uid`** sur un binaire (dans le même genre, on peut déléguer des permissions avec la fameuse commande **sudo**) afin de permettre à l'utilisateur d'accéder à la ressource mais cela n'est pas transparent et rajoute un niveau de complexité propre à décourager sur le long terme même le plus motivé des administrateurs.

Tous prosélystes que nous sommes, il nous faut donc bien admettre que la gestion en tout ou rien des permissions en vigueur sous Linux (hérité du monde Unix) est certes simple à appréhender (quelques exemples permettent de comprendre rapidement comment cela fonctionne), mais reste difficile à étendre et à faire évoluer : La demi-mesure n'étant pas de mise, les scénarios complexes sont dès lors difficilement implémentables (le problème devient même quasi-insoluble lorsque l'on souhaite non pas donner accès mais bien exclure un utilisateur des droits sur une ressource au sein d'un même groupe !).

Pourtant, peu de gens savent qu'il est pourtant désormais possible sous Linux d'accéder à une gestion plus fine des droits d'accès en utilisant les listes de contrôle d'accès, les fameuses **ACL** (*Access Control List*) [1].

Introduction aux listes de contrôle d'accès

Il était temps. La majorité des Unix propriétaires (**AIX**, **IRIX**, **SOLARIS**, **TRU64**, **HP-UX** et même **UnixWare**), **FreeBSD**, **Windows** (NT/2000/XP uniquement à travers NTFS), **Novell Netware** et même **MaxOS X** (depuis la version TIGER 10.4) supportent les **Listes de Contrôle d'accès** (**ACL** en anglais, pour *Access Control List*).

Même des logiciels comme **SQUID** ont implémenté le concept des ACL dans leur configuration afin de gérer les droits d'accès des utilisateurs. On voit que l'idée a fait son chemin.

Linux est longtemps resté dans le domaine le vilain petit canard et il aura fallu la sortie officielle du noyau 2.6 pour voir cette fonctionnalité disponible en standard dans le noyau Linux (plus précisement, à partir de la version 2.5.46 même si la fonctionnalité a été rajoutée dans les dernières versions de la branche 2.4 et était disponible auparavant sous forme de patches).

Si la vie était bien faite, il n'y aurait qu'une version des ACL. Cela n'est pas le cas et il faut se reporter aux groupes de travail POSIX sur les normes **1003.1e** et **1003.2c** pour trouver traces d'une tentative de normalisation dans le domaine. Ces groupes de travail ont en fait été actifs pendant 13 ans mais, pour différentes raisons, les projets de standard correspondants ont été abandonnés et les groupes de travail dissous en 1998. La dernière version de travail de la spécification a été rendu publique en 1997 (draft 17) et est disponible sur [2].

Tout n'a pas été perdu toutefois car la majorité des implémentations sont basées sur ces propositions à l'exception notables de HP-UX, Netware et de Windows qui ont suivis des directions différentes. On ne sera donc pas dépaysé en passant d'un système à l'autre dans la plupart des cas (le support sera en tout cas parfait entre les différentes distributions Linux).

Les principaux systèmes de fichiers disponibles sous Linux supportent en standard les ACL ce qui explique pourquoi on peut massivement les utiliser : **Ext2/3**, **JFS**, **ReiserFS** et **XFS** (XFS tient une place à part car la fonctionnalité est activée par défaut alors qu'il faut préciser l'option **acl** lors du montage des autres types de partition pour y avoir droit). Le système de fichier **NFS** nécessite des patches qui ne sont peut-être pas disponibles en standard dans votre distribution.

On peut facilement valider le support des ACL en tapant la commande suivante dans une console (si la commande existe[1] ou /boot/config*) :

```
# zgrep POSIX_ACL /proc/config.gz
CONFIG_EXT2_FS_POSIX_ACL=y
CONFIG_EXT3_FS_POSIX_ACL=y
CONFIG_REISERFS_FS_POSIX_ACL=y
CONFIG_JFS_POSIX_ACL=y
CONFIG_FS_POSIX_ACL=y
CONFIG_XFS_POSIX_ACL=y
```

Mais avant que débute notre voyage, il nous faut d'abord nous initier à une fonctionnalité peu connue (donc peu utilisée) qui pourtant est la base techniques des ACL sous Linux, les **Attributs Etendus** (**EA** en anglais, pour *Extended Attributes*).

Petit détour par les Attributs Etendus

Prenez l'exemple d'un fichier MP3 : N'importe quel utilisateur est capable, avec le bon logiciel, d'associer un album, un titre et un artiste à un fichier MP3 au travers de l'extension **ID3** [3] (c'est le cas avec le logiciel **kid3** sous KDE). C'est ce que l'on appelle des méta-données, au sens où ce sont des données qui décrivent d'autres données. On retrouve la même approche dans l'en-tête **Exif** [4] des fichiers JPEG issus d'un appareil photo : On récupère la date et l'heure de la photo ainsi que différentes informations techniques sur le contexte de la prise de vue (temps d'exposition, ouverture, lumière, ...).

C'est en soit bien pratique mais ces données sont incorporées au fichier d'image (elles sont généralement présentes à la fin du fichier) ce qui n'est pas forcément le plus élégant.

En suivant le même raisonnement, avouez qu'il serait tout aussi pratique de pouvoir associer des données arbitraires à un fichier sans altérer son contenu. Imaginez à quel point cela serait utile pour annoter ou versionner vos documents, préciser l'auteur voire ajouter un libellé descriptif pour préciser le contenu, et cela sans altérer le contenu réel du fichier.

Arrêtez de rêvasser car cela existe maintenant et tout de suite sous Linux en utilisant la puissance et la simplicité des **attributs étendus**.

---

1 Cette fonctionnalité peut-être désactivée parce qu'elle consomme inutilement de la mémoire

Vous pouvez vérifier tout de suite si votre distribution supporte en l'état les attributs étendus en tapant la commande suivante dans une console (si la commande existe ou /boot/config*) :

```
# zgrep ATTR /proc/config.gz
CONFIG_EXT2_FS_XATTR=y
CONFIG_EXT3_FS_XATTR=y
CONFIG_REISERFS_FS_XATTR=y
CONFIG_DEVPTS_FS_XATTR=y
CONFIG_CIFS_XATTR=y
```

Ce qui est sans doute le cas si votre distribution supporte aussi les ACL car les attributs étendus sont indispensables au bon fonctionnement des ACL. Une liste de contrôle d'accès est en fait tout simplement mémorisée dans les attributs étendus d'un fichier : Les ACL sont tout simplement un type particulier d'attributs étendus (il en existe actuellement trois sortes comme nous le verrons plus loin).

Un attribut étendu est un couple libre de la forme **variable=libellé**. La variable (tout comme dans un shell) permet d'identifier l'attribut (de façon unique sur le fichier) alors que le libellé est une chaîne de caractère complètement arbitraire qui précise le contenu utile de la variable.

On peut associer un ou plusieurs attributs étendus à n'importe quel type de fichier : Fichier normal, répertoire, lien et fichiers spéciaux.

Si vous utilisez un système de fichiers autre que XFS, il vous faudra activer l'option **`user_xattr`** lors du montage du système de fichiers. Vérifiez dans votre fichier **/etc/fstab**, cela est peut-être déjà mis en place.

Comme nous l'avons déjà précisé plus haut, il existe trois classes d'attributs étendus :

Classe	description
`user.*`	Cette classe est réservée aux utilisateurs pour y saisir de façon arbitraire leurs méta-données
`system.*`	Cette classe est actuellement réservée par les ACL pour y stocker les listes de contrôles d'accès comme nous le verrons plus loin dans l'article
`trusted.*`	Cette classe est réservée pour un usage futur (non supportée actuellement)

En ligne de commande, nous pouvons accéder aux attributs étendus de la classe **user** par l'intermédiaire des commandes **setfattr** et **getfattr** (une commande **attr** est disponible mais elle est uniquement présente par soucis de compatibilité avec le système d'exploitation IRIX de SGI).

La commande **getfattr** permet de consulter les attributs étendus d'un fichier alors que la commande **setfattr** permet de supprimer, ajouter ou modifier un attribut étendu.

Prenons un exemple simple. Je souhaite organiser mes documents de façon plus rationnelle en leurs attribuant des informations qui me serviront plus tard lors de recherches ultérieures (on préfixe chaque attribut avec la classe réservée **`user.*`**).

```
setfattr -n user.auteur -v "Nom Prenom" article_acl.odt
setfattr -n user.type -v "article" article_acl.odt
setfattr -n user.article.titre -v "Gestion des  ACL sous Linux" article_acl.odt
setfattr -n user.article.type -v OpenDocument article_acl.odt
setfattr -n user.mots_clef -v "EA,ACL,LINUX" article_acl.odt
```

Comme nous venons de le voir, l'utilisation judicieuse du point permet d'organiser de façon intuitive vos attributs en domaines et sous-domaines (un espace de nommage en somme) ; plutôt pratique lorsque le nombre d'attributs que l'on souhaite déclarer est potentiellement élevé. C'est préférable car cela structure et harmonise la déclaration des attributs étendus (de toute façon rien n'est imposé).

Pour consulter les attributs étendus d'un fichier, il faut saisir la commande suivante :

```
# getfattr -n user.article.type article_acl.odt
user.article.type="OpenDocument"
```

Si vous souhaitez uniquement récupérer la valeur de l'attribut étendu :

```
# getfattr --only-values -n user.article.type article_acl.odt
OpenDocument
```

Si vous souhaitez récupérer tous les attributs de type **user.article** :

```
# getfattr -m user.article article_acl.odt
user.article.titre
user.article.type
```

Pour finir, la commande suivante permet de supprimer un attribut étendu :

```
# setfattr -x user.article.type
```

Plutôt simple et intuitif !

Cette disgression nous ramène tout naturellement vers la gestion des listes de contrôle d'accès qui seront stockés au moyen des attributs étendus sans risque aucun d'altération du contenu du fichier.

Gestion avancée des Liste de contrôle d'accès

On utilise les commandes **getfacl** et **setfacl** pour consulter puis modifier les ACL sur un fichier.

Les ACL se divisent en fait en deux classes principales, l'**ACL minimale** et l'**ACL étendue** ; notions auxquelles il faut rajouter celle d'**ACL par défaut** (qui ne va s'appliquer qu'aux répertoires).

L'ACL minimale est exclusivement composée d'élements de type « **Propriétaire** » (*owner*), « **Groupe** » (*owning group*) et « **Autres** » (*other*) et correspond à la traduction littérale du modèle de gestion des droits Unix traditionnels que nous avons abordés au début de l'article : Les ACL ne remplacent pas mais étendent le modèle standard afin de préserver la compatibilité avec l'existant.

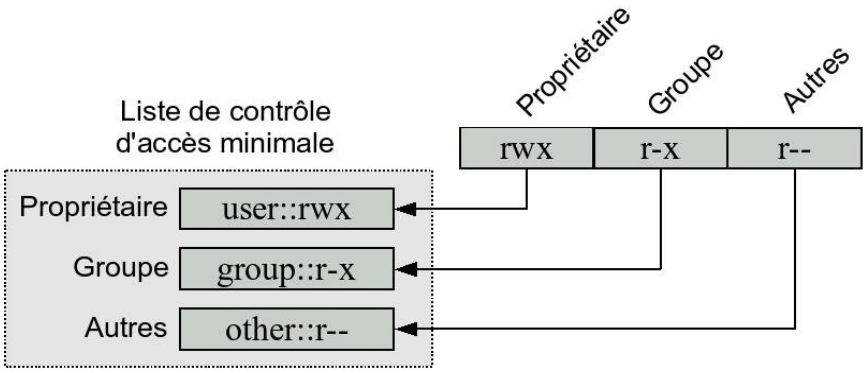

Figure 1 : *Liste de contrôle d'accès minimale*

Une ACL étendue prolonge les droits de l'ACL minimale et introduit la notion de « **Masque** » (*mask*), d' « **Utilisateurs nommés** » (*named user*) et de « **Groupes nommés** » (*named group*).

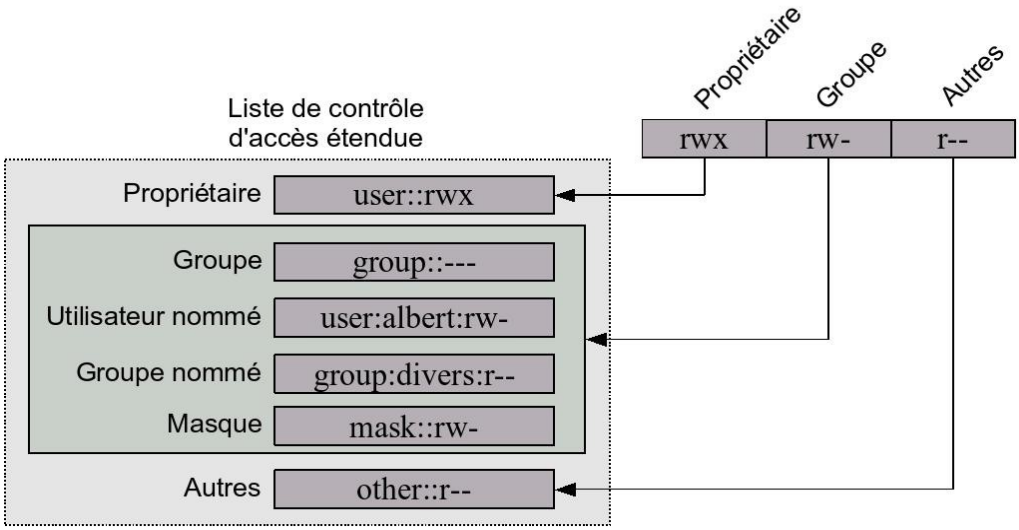

Figure 2 : Liste de contrôle d'accès etendue

Il n'est pas difficile de comprendre le rôle des « Utilisateurs nommés » et des « Groupes nommés » qui vont nous permettre d'attribuer individuellement des droits à, respectivement, des utilisateurs et des groupes.

Prenons un exemple concret. Nous disposons d'un fichier **exemple.txt** qui est configuré avec les droits **600**

(rw-------) pour l'utilisateur **lionel:users**. Toute tentative de lire le fichier avec un utilisateur **test** appartenant exclusivement au groupe **test** abouti logiquement à un échec :

```
# id
uid=1003(test) gid=1000(test)
# ls -l exemple.txt
-rw-------  1 lionel users 6492 2004-12-29 19:36 exemple.txt
# cat exemple.txt
cat: exemple.txt: Permission non accordée
```

Si on exécute la commande **getfacl** avec le fichier en argument, on obtient l'ACL minimale suivante :

```
# getfacl exemple.txt
file: exemple.txt
owner: lionel
group: users
user::rw-
group::---
other::---
```

On souhaite maintenant donner accès au fichier en lecture et écriture uniquement pour l'utilisateur test (nous allons étendre l'ACL) au moyen de la commande **setfacl** :

```
# setfacl -m user:test:rw exemple.txt
# ls -la exemple.txt
-rw-rw----+ 1 lionel users 6492 2004-12-29 19:36 exemple.txt
# getfacl exemple.txt
...
user::rw-
user:test:rw-
group::---
mask::rw-
other::---
```

On s'aperçoit qu'un caractère supplémentaire **+** à fait son apparition lorsque l'on consulte les droits du fichier de façon classique (cela n'est pas forcément une règle partagée par toutes les distributions, mais c'est le cas actuellement sur la SuSE).

On voit aussi apparaître la notion de « Masque » dans la liste des ACL étendues : C'est en effet le mécanisme qui est à l'origine de la façon dont les droits d'accès vont être perçus par les applications qui ne prennent pas en charge les ACL.

Au sein d'une ACL, les droits définis par les éléments « Propriétaire » et « Autres » sont conservés en l'état et définissent respectivement les classes « Propriétaire » et « Autres » du fichier (on se place du point de vue des droits Unix) ; ce qui n'est pas le cas pour tous les autres éléments (Groupe, Utilisateur nommé et Groupe nommé) qui n'ont pas tous d'équivalents dans le monde Unix et qui seront donc agrégés puis filtrés par le masque de l'ACL pour constituer la classe « Groupe » : Si des droits sont disponibles aussi bien dans l'un de ces trois éléments que dans le masque, ils deviennent actifs (par symétrie, les éléments qui ne sont pas inclus dans le masque ne sont pas actifs).

Type		**Mise en forme**	**Version abrégée**
Propriétaire	*Owner*	user::rwx	u::rwx
Groupe	*Owning group*	group::rwx	g::rwx
Autres	*Other*	other::rwx	o::rwx
Utilisateur nommé	*Named user*	user:[nom]:rwx	u:[nom]:rwx
Groupe nommé	*Named group*	group:[nom]:rwx	g:[nom]:rwx
Masque	*Mask*	mask::rwx	m::rwx

***Figure 3** : Liste des différents types d'éléments ACL*

Le masque n'est pas modifiable par l'utilisateur et s'adapte automatiquement aux modifications des entrées ACL ; si on supprime le droit d'écriture sur le groupe, on obtient le résultat suivant (seul le masque sera modifié) :

```
# chmod g-w exemple.txt
# getfacl exemple.txt
...
user::rw-
user:test:rw-                   #effective:r--
group::---
mask::r--
other::---
# ls -l exemple.txt
-rw-r-----+ 1 lionel users 6492 2004-12-29 19:36 exemple.txt
```

Pour supprimer une ACL :

```
# setfacl -x user:test exemple.txt
```

Gérer les ACL n'est pas très compliqué si l'on a bien assimilé la notion d'ACL minimale, d'ACL étendue et le concept de masque ; il nous reste à aborder la notion d'ACL par défaut et vous saurez tout ce qu'il faut savoir pour manipuler des ACL.

Une ACL par défaut s'applique exclusivement à un répertoire et spécifie les droits d'accès dont les différents fichiers de ce répertoire (fichiers ou répertoires) héritent lorsqu'ils sont créés.

La similitude n'est pas accidentelle avec la commande **umask** qui permet de définir les droits d'accès lors de la création d'un fichier car si le répertoire parent ne possède aucune ACL par défaut, on applique en effet les droits définis par la commande umask ; sinon, on applique les droits définis par l'ACL par défaut et on ne tient alors pas compte de la valeur de la commande umask.

Les règles qui s'appliquent lors de la création d'un fichier ou d'un sous-répertoire (dans un répertoire qui possède une ACL par défaut) sont les suivantes :

- Un fichier utilise l'ACL par défaut du répertoire parent pour constituer sa propre ACL d'accès,
- Un sous-répertoire utilise aussi l'ACL par défaut du répertoire parent pour son ACL d'accès mais hérite surtout de l'ACL par défaut du père pour constituer sa propre ACL par défaut (une ACL par défaut se duplique de répertoire en sous-répertoire).

Par exemple, si j'autorise l'utilisateur **test** à venir créer des fichiers dans le répertoire **Documents/** de l'utilisateur **lionel** :

```
# setfacl -m user:test:rwx Documents/
# setfacl -dm user:test:rwx Documents/
# getfacl Documents/
...
user::rwx
user:test:rwx
group::r-x
mask::rwx
other::r-x
default:user::rwx
default:user:test:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
```

Exemple de création d'un répertoire :

```
# cd Documents/
# mkdir repertoire
# ls -l repertoire
drwxrwxr-x+ 1 test test 0 2005-10-23 17:55 repertoire
```

Exemple de création d'un fichier (on remarquera que le droit d'exécution n'est pas positionné, et cela quelque soit le masque) :

```
# touch fichier
# ls -l fichier
-rw-rw-r--+ 1 test test 0 2005-10-23 18:02 fichier
```

Persistance des Attributs Etendus et des ACL

Même si les principales commandes de type **cp** (avec l'option **-p**) ou **mv** prennent désormais en charge les ACL, cela n'est pas le cas actuellement des navigateurs de fichiers (**Nautilus**, **Konqueror**, ...) ou, pire, des utilitaires d'archivage les plus utilisés sous Linux (**tar**, **dump**, ...).

Par exemple, lors de la copie d'un fichier avec Konqueror, les attributs du fichier d'origine sont « oubliées » au passage ; de plus, il n'est à ce jour possible d'éditer de façon conviviale et graphique les droits ACL (exemple de KDE ou GNOME).

Cependant, les choses sont en train de bouger après plusieurs années de stagnation et l'on s'attend à ce que la gestion des ACL soit officiellement prisent en compte à partir de la version 3.5 de KDE [5] ; même agitation coté GNOME où les choses viennent afin de redémarrer dernièrement [6].

Les applications de sauvegarde standard ne mémorisant pas les ACL, il faut alors se diriger vers **rdiff-backup** qui supporte désormais les ACL [7] et l'utilitaire **star** [8] qui garantie la prise en charge complète des ACL à travers le format de sauvegarde **PAX** :

```
# star -Hexustar -acl -c f=documents.star Documents/
# star -acl -x f=documents.star
```

Une autre solution consiste à sauvegarder soit même les attributs ACL par la commande **getfacl** :

```
# tar czvf documents.tgz Documents/
# getfacl -R --absolute-names Documents/ > documents.acl
# rm -rf  Documents/
# tar xzvf documents.tgz
# setfacl --restore=documents.acl
```

Une mention spéciale au système de fichier XFS qui a su très tôt gérer les ACL : On a déjà remarqué que le support des ACL (et donc des Attributs Etendus) était activé en standard par ce système de fichiers mais il existe aussi un jeu de commandes spécifiques, **xfsdump** et **xfsrestore**, qui permettent de conserver ce type d'attributs (cependant, c'est exclusivement réservé aux systèmes de fichier XFS et donc à n'utiliser que dans un environnement strictement homogène).

Pour finir

Le concept introduit par les attributs étendus est très prometteur. On pense presque immédiatement aux bureaux Linux (KDE, GNOME, ...) qui gagneraient en convivialité si ils pouvaient supporter ce type de technologie : Pouvoir associer des états synthétiques à des fichiers, permettre la saisie et l'affichage d'informations contextuelles, faciliter le travail d'indexation des données, ... serait un vrai plus réellement apprécié par les utilisateurs.

De leur coté, la beauté des listes de contrôle d'accès réside dans leur parfaite compatibilité avec le système de droits Unix traditionnels (les applications qui supportent ou non les ACL cohabitent parfaitement entre elles), ce qui autorise la mise en oeuvre de scénarios très complexes sans perturber le fonctionnement des applications existantes. Tout le monde n'a pas forcément besoin de la puissance des ACL.

Concernant justement les ACL, le nombre d'entrées maximum supporté par fichier est un critère important dans le choix du système de fichiers où l'on souhaite les mettre en place : Ils sont respectivement de 25 entrées pour XFS, 32 pour Ext2/3 ou enfin 8191 pour ReiserFS et JFS.

Comment intégrer la prise en compte des attributs étendus ou des listes de contrôle d'accès dans vos programmes ? Pas d'affolement, le lien [10] vous donne toutes les indications utiles (c'est assez simple, en fait) afin de consulter ou préserver ce type d'attributs (des librairies et de nombreux exemples sont mis à disposition).

Si vous souhaitez aller plus loin, vous pouvez consulter le document [11] de Andreas Gruenbacher (SuSE), l'un des gourous de la mise en oeuvre de ACL sous Linux (on lui doit un grand merci, ainsi qu'aux ingénieurs du système de fichier XFS de chez Silicon Graphic).

Références

[1] Linux ACL Homepage : http://acl.bestbits.at/

[2] Normes POSIX 1003.1e et 1003.2c : http://wt.xpilot.org/publications/posix.1e/download.html

[3] Tag ID3 : http://fr.wikipedia.org/wiki/ID3

[4] Format Exif : http://fr.wikipedia.org/wiki/EXIF
[5] ACL sous KDE : http://bugs.kde.org/show_bug.cgi?id=6976
[6] ACL sous GNOME : http://bugzilla.gnome.org/show_bug.cgi?id=62835
[7] Utilitaire rdiff-backup : http://www.nongnu.org/rdiff-backup/
[9] Archiveur star : ftp://ftp.berlios.de/pub/star/
[10] Intégrer les EA et les ACL dans vos programmes : http://www.suse.de/~agruen/ea-acl-copy/
[11] POSIX Access Control Lists on Linux : http://www.suse.de/~agruen/acl/linux-acls/online/