04 - Système de fichier

Gaétan Richard
gaetan.richard@info.unicaen.fr

L2 – S4 2011/2012

1. Rappels

Utilisateurs et groupes

Chaque utilisateur UNIX possède un identifiant (**uid**).

Il existe également des identifiants de groupe (**gid**)

Chaque utilisateur peut appartenir à un ou plusieurs groupes.

2. Présentation

Hiérarchie

Le système de fichier POSIX est organisé en une arborescence unique.

Le séparateur de dossier est le symbole slash (**/**).

La **racine** du système de fichier est dénoté par la répertoire **/**.

Adressage

Il existe deux façon de parler d'un objet dans le système de fichiers :

- de façon absolue en donnant sa position dans la hiérarchie (ex : `/Users/grichard/Enseignements/2010-2011`);
- de façon relative en donnant sa position par rapport au répertoire courant (ex : `sand/log`, `../TD`).

`.` et `..`

Dans le modèle POSIX, tout répertoire possède obligatoirement deux sous-répertoires particuliers : `.` et `..`

- `.` désigne le répertoire lui-même ;
- `..` désigne le répertoire parent.

Point de montage

L'arborescence est obtenue en montant des supports physiques à un emplacement précis.

Cette opération est réalisée à l'aide de la commande **mount**.

Il est possible d'effectuer des montages / démontages à chaud.

De ce fait, les objets du système de fichiers peuvent être situés sur des supports physiques différents. Ceci se fait de façon complètement transparente vis-à-vis de l'utilisateur.

Fichiers et dossiers

Les **fichiers** (réguliers) contiennent les données.

Les **dossiers** forment l'arborescence du système et sont organisés sous forme d'arbre. Leur donnée est la liste des fichiers qu'ils contiennent.

Lors de l'utilisation de **ls -l** les dossiers sont reconnaissables à la présence d'un **d** au début de la ligne.

Liens symbolique

Les **liens symboliques** sont des liens vers un fichier ou un dossier situé ailleurs dans le système de fichier.

Le lien peut être décrit de façon relative ou absolue.

Les liens symboliques sont préfixés par **l** lors de l'utilisation de **ls -l**. De plus, l'objet pointé est affiché.

Fichiers spéciaux

Les périphériques sont représentés par des **fichiers spéciaux**. Ces fichiers peuvent être de type caractère ou blocs.

Il sont préfixés par **c** ou **b** lors de l'utilisation de **ls -l**

Il en existe certains d'utilisation courante :

- **/dev/null** qui est un *trou noir* ;
- **/dev/zero** qui contient une infinité de 0 ;
- **/dev/random** et **/dev/urandom** qui contiennent des données aléatoires.

Tubes

Les **tubes** (pipe) sont des mécanismes gérant une file d'entrées / sorties.

Ils sont reconnaissable au préfixe **p** lors de l'utilisation de **ls -l**.

Socket

Les **sockets** permettent de communiquer au travers du réseau.

Elles ne sont pas visibles dans l'arborescence mais se manipulent de la même façon que les fichiers.

3. Inoeud

Métadonnées

Selon son type, le fichier peut contenir des données.

De plus, on associe à chaque fichier un ensemble de **métadonnées** :

- type de fichier ;
- propriétaire ;
- droits d'accès ;
- taille ;
- dates de création / modification ;
- …

Sous UNIX, ces données sont traditionnellement stockés dans la structure d'**inoeud** (inode).

Exemple

Dossier :

N° 123
type :dossier
proprio : grichard
droits : rwxrwxr-x
création : 08/09/10 11 :12
...

. -> 123
.. -> 110
toto.tex -> 126
sand -> 165
...

Fichier :

N° 126
type : fichier
proprio : grichard
droits : rw-r–r–
création : 08/09/10 12 :34
...

\documentclass{article}

\usepackage{xltxrta}
% un package utile
...

Important : la notion de nom n'existe qu'au travers des répertoires.

Compteur de lien

Parmi les métadonnées, il existe un **compteur de lien** qui indique combien d'entrées de répertoire pointent vers un fichier.

Pour les éléments autres que les dossiers, il est possible de créer une autre entrée de dossier pointant vers le fichiers (link) ce qui incrémente le compteur.

Lorsqu'une entrée est supprimée (unlink) , on décrémente le compteur.

Un fichier existe tant que le compteur est non nul.

Organisation sur le disque

Le disque physique est découpé en blocs inode et data.

Les inoeuds pointent vers les données associées.

Le système d'exploitation garantit l'intégrité du système de fichier :

- Pas de liens multiples vers un dossier ;
- Gestion des blocs libres ;
- Récupération de l'espace libéré par l'effacement ;
- Réparation des incohérences ;
- …

Système de fichiers

Il existe de nombreux systèmes de fichiers différents ayant chacun leurs limites, leurs avantages et leurs inconvénients : ext2, FAT32, NTFS, ReiserFS, XFS, HFS plus, TrueFFS, CDFS (ISO 9660) …

Caractéristiques remarquables :

- Sensible/Insensible à la casse ;
- Présence de journalisation ;
- Gestion des petits fichiers ;
- Vitesse d'accès / d'écitures ;
- Redondance des données ;
- Limites
- …

Stat

Pour obtenir les métadonnées d'un fichier, il est souvent suffisant d'utiliser l'option **-l** de la commande **ls**.

Si l'on veut des détails plus précis, il reste toujours la commande **stat**.

Exemple :

```
$ stat sand
234881026 15064753 drwxr-xr-x 20 grichard (140) 0 680
"Feb  7 10 :15 :19 2010" "Feb  7 10 :21 :05 2010"
"Feb  7 10 :21 :05 2010" "Jan 24 16 :56 :50 2010"
4096 0 0 sand
$
```

Stat / fstat

Il existe les commandes :

- **os.stat** pour obtenir les métadonnées d'un fichier ;
- **os.statvfs** pour celles du point de montage correspondant.

Exemple :

```
>>> import os
>>> os.stat(".")
posix.stat_result(st_mode=16877, st_ino=768967L, st_dev=234881026L,
st_nlink=68, st_uid=4064, st_gid=140, st_size=2312L,
st_atime=1265546689, st_mtime=1265546692, st_ctime=1265546692)
>>> os.statvfs(".")
posix.statvfs_result(f_bsize=1048576, f_frsize=4096,
f_blocks=34045952L, f_bfree=20836713L, f_bavail=20772713L,
f_files=34045950L, f_ffree=20772713L, f_favail=20772713L,
f_flag=0, f_namemax=255)
>>>
```

Utilisateur / Groupe

Il est possible de modifier le propriétaire / groupe dans les métadonnées d'un fichier en utilisant :

- la commande **chown** dans le shell ;
- la fonction **os.chown** dans python.

Exemples :

```
$ chown grichard :140 sand
$
```

```
>>> os.chown(".",0,140)
Traceback (most recent call last) :
  File "<stdin>", line 1, in <module>
OSError : [Errno 1] Operation not permitted : '.'
```

Droits (présentations)

Chaque fichier possède des métadonnées sur les droits.

Il existe 3 types de droits :

- **r** le droit de lecture (voir le contenu dans le cas de dossier) ;
- **w** le droit d'écriture (ajouter / supprimer des fichier dans le cas d'un dossier) ;
- **x** le droit d'exécution (permet de traverser les dossiers).

Ces droits sont définis :

- pour le propriétaire (**u**) ;
- pour les membres du groupe (**g**) ;
- pour les autres utilisateurs (**o**).

Droits (principes)

Les droits d'accès sont pris en compte depuis la racine du système de fichiers (*i.e.*, pour accéder à un dossier, il faut avoir le droit d'accéder au dossier parent).

En particulier, dans le cas de liens symboliques, il faut avoir les droits d'accès au lien et à la cible.

Attention à l'ordre de prise en compte des droits.

Droits (notations)

On note usuellement les droits en mettant à la suite ceux du propriétaire, suivi du groupe et des autres utilisateurs.

Pour chacun de ces droits, on donne les lettres correspondantes aux droits dans l'ordre **rwx** en remplaçant par un tiret (**-**) si le droit n'est pas présent.

Exemple :

```
$ ls -l .
-rwxr--r--  1 grichard  140   51 25 jan 09 :18 script.py~
lrwxr-xr-x  1 grichard  140   19 24 jan 17 :05 seq -> /opt/local/bin/gseq
-rw-r--r--  1 grichard  140  642  7 fév 10 :21 texput.log
prw-r--r--  1 grichard  140    0  7 fév 10 :15 toto
```

Droit (notation octale)

Il existe également une notation qui assimile chaque triplet **rwx** à un entier en le regardant comme un nombre en binaire :

r	**w**	**x**
4	2	1

Il est alors possible de représenter les droits d'un fichier par un nombre octal de trois chiffres. (ex : **755** pour représenter **rwxr-xr-x**).

Important : il s'agit d'un entier écrit en octal et non pas de l'entier 755. En python, un tel entier s'écrit en ajoutant un 0 initial.

Chmod

Pour modifier les droits, il suffit d'utiliser la commande **chmod**.

Cette commande peut s'utiliser :

- En appelant les modifications par leur lettres (ex : `u+r`, `go-w`, `ug+rw`, ...) ;
- la lettre **a** peut être utilisée à la place de ugo ;
- En utilisant la notation octale.

Exemples :

```
$ chmod u+x truc
$ chmod 755 truc
$
```

Chmod

En python, il suffit d'utiliser l'appel système **os.chmod**.

Note : Il est possible d'utiliser la notation octale pour le mode ou les constantes définies dans le module stat.

Exemple :

```
>>> os.chmod("truc",0755)
>>> os.chmod("truc",stat.S_IRUSR | stat.S_IWUSR)
```

Droits (autres droits)

Il existe trois autres droits :

- **suid** (**u+s**) : le programme lancé acquière en plus les droits du propriétaire du fichier.
- **sgid** (**g+s**) : même chose que suid avec les droit du groupe. Pour un dossier, les fichiers créés dans ce dossier seront du groupe du dossier.
- **sticky bit** (**a+t**) : limite le droit de suppression de fichiers à leur seul propriétaires.

Notation : Ces bits sont indiqués à la place du droit d'exécution de l'entité correspondante (**u,g,o**). Si le bit d'exécution n'est pas présent, cette valeur est parfois affichée en majuscule.

Notation octale : Pour la notation octale, on ajoute un chiffre au début du mode (4 pour suid, 2 pour sgid, 1 pour t).

Exemples

Quelques exemples

ACL

Il existe un mécanisme plus fin de gestion des droits au travers des **ACL** (Access Control List).

Il est possible de les utiliser au travers des appels **getfacl** et **setfacl**.

Note : Ces méthodes sont très récentes et peu utilisées.

4. Primitives

Appels système

Pour réaliser les opérations élémentaires précédentes, il faut interagir avec le système d'exploitation. Ceci se fait par l'intermédiaire de fonction particulières appelées **appels systèmes** (ex : **chmod**).

Lors de l'appel de ces fonctions, c'est le système qui prend la main et renvoie le résultat.

Ces fonctions sont également appelée **primitives systèmes** car elles sont presque toujours des opérations élémentaires qui nécessitent de dialoguer avec le système d'exploitation.

Un autre point important pour la suite est que les appels systèmes sont **atomiques**.

Retour sur les descripteurs de fichiers

Pour pouvoir travailler sur des fichiers, il faut d'abord les ouvrir pour obtenir un descripteur de fichier.

Il est important de noter que le descripteur de fichier contient également des droits qui peuvent être des restrictions des droits que l'utilisateur a sur le fichier.

De nombreuses fonctions déjà rencontrées peuvent également s'appliquer directement à des descripteurs de fichiers en utilisant une variante préfixée par `l` (ex : **lchmod**, …)

En shell

En shell, la plupart des manipulations sur le contenu des fichiers se fait par l'intermédiaire des redirections (**<**, **<<**, **>** et **>>**)

Vous pouvez vous référer au cours 2 pour plus de détails.

Note : lors de la création de fichier ou de dossier, les droits utilisés dépendent du **umask**.

Open / Close

Python dispose des fonctions **open**. Cette fonction sert à convertir un fichier en un descripteur de fichier.

Il est possible de spécifier le mode avec en particulier :

- **r** lecture,
- **w** écriture,
- **a** append,
- **r+**, **w+** lecture/écriture.

Pour fermer le descripteur, il suffit d'appeler la méthode **close** de cet objet.

Il existe également la construction :
`with open("hello.txt") as f :`
qui permet d'automatiser la fermeture du fichier.

Lecture / écriture

Pour lire / écrire, il suffit d'utiliser les méthodes présentes dans l'objet "file descriptor" :

- **read()** : renvoie le contenu du fichier ;
- **readline()** : renvoie une ligne du fichier ;
- **write(*s*)** : écrit la chaîne *s* dans le fichier ;

Note : Il est également possible d'utiliser la construction :
`for line in f :` qui itère sur les lignes du fichier.

Flush

Pour des raisons d'efficacité, les écritures sont mises en mémoire tampon et réellement écrites lorsqu'il y en a suffisamment ou que l'on ferme le descripteur.

Il est possible de force l'écriture à l'aide des fonctions **os.fsync(*fd*)** et de la méthode **flush()** associée au descripteur de fichier.

Seek

Il est aussi possible de se déplacer dans un fichier à l'aide des méthodes fournies par l'objet "file descriptor" :

- **tell()** : indique la position courante ;
- **seek(*off*,0)** se place à la position *off* mesuré depuis le début du fichier ;
- **seek(*off*,1)** se déplace de *off* dans le fichier (déplacement relatif) ;
- **seek(*off*,2)** se place à la position *off* par rapport à la fin du fichier.

Un exemple complet

```
>>> f = open(”test”,”r+”)
>>> print f
<open file ’test’, mode ’r+’ at 0x708e0>
>>> f.write(”toto”)
>>> f.write(”tata”)
>>> f.seek(0,0)
>>> f.readline()
’tototata’
>>> f.tell()
8L
>>> f.write(”\ntor\n”)
>>> f.tell()
13L
>>> f.seek(-4,1)
>>> f.tell()
9L
>>> f.readline()
’tor\n’
>>> f.close()
```

Créer des liens

Il est possible de créer un lien physique (c'est-à-dire de rajouter une entrée dans un répertoire vers l'inoeud un fichier existant (autre que répertoire).
Il suffit d'utiliser la commande

ln *fichier_pointé nom de la nouvelle entrée*

Le compteur de lien du fichier est donc augmenté.

En ajoutant l'option **-s**, on crée un liens symbolique.

Et en python

Python dispose des primitives : **os.link** et **os.symlink** pour créer des liens physiques et symboliques.

Leur utilisation et fonctionnement est le même que les commandes shell vues précédemment.

Liens symboliques et métadonnées

Dans le cas des liens symboliques, toutes les demandes de métadonnées peuvent être comprises comme portant soit sur le fichier pointé par le lien, soit sur le lien lui-même.

Généralement, les fonctions modifient / renvoient les métadonnées du fichier pointé mais il existe des versions différentes (préfixés) par `l` qui agissent sur le lien lui-même. Dans le cas des programmes, cette dualité prend souvent l'aspect d'une option (ex : **chmod -h** ou **stat -L**).

On notera que modifier les métadonnées d'un lien symbolique est rarement une bonne idée. Le comportement des droits n'étant pas spécifié.

Création / Suppression

Il est possible de créer / supprimer des répertoires à l'aide :

- des commandes **mkdir** et **rmdir** du shell ;
- des fonctions **os.mkdir** et **os.rmdir** de python.

Pour pouvoir supprimer un répertoire, il faut que celui-ci soit vide.

Chdir / getcwd

Pour changer le répertoire courant, il existe la commande **cd** du shell. Son équivalent python est la commande **os.chdir** qui fonctionne exactement de la même façon.

Il est possible (même si cela ne fait pas partie des fonctions originale de POSIX) de connaître le répertoire courant en utilisant la commande **pwd** du shell (ou la variable **$PWD**). Son équivalent python est la fonction **getcwd**.

listage du contenu

En shell, il est très facile de lister le contenu d'un répertoire à l'aide de la commande **ls**.

En python, la fonction **os.listdir(*path*)** retourne la liste des entrée d'un dossier en omettant `.` et `..`

Création

Les tubes se créent à l'aide de la commande **mkfifo** en shell ou bien **os.mkfifo** en python.

Le pipe | vu dans les commandes crée un tube anonyme. Sous python, il est possible d'utiliser la fonction **os.pipe()** pour en créer un.

L'utilisation se fait exactement de a même façon qu'un fichier régulier.

Manipulation

Rappel : Les noms de fichiers n'existent qu'au travers de répertoires.

Suppression

La suppression à l'aide de la commande **rm** qui utilise l'appel système **unlink**.
Attention : la suppression est irréversible.

En python, la suppression est disponible par l'intermédiaire de la fonction **os.unlink** (également appelée **os.remove**).

Copie

Il n'y a pas d'appel système pour la copie, il faut prendre le contenu d'un fichier et l'écrire dans un autre fichier à la main.

Dans le cas du shell, la commande **cp** automatise cette opération.

rename / move

Pour déplacer un fichier, il suffit de faire un **link** suivi d'un **unlink**.

Problème 1 : Cela ne marche pas pour les dossiers, on introduit donc l'appel système **rename**.

Problème 2 : La commande **rename** ne marche pas si la source et la destination ne sont pas sur le même système de fichier, il est indispensable de faire une copie suivie d'un effacement.

rename / move

La commande **mv** du shell effectue en fonction du contexte :

- soit un rename ;
- soit une copie suivi d'une supression.

Fichier temporaire

Pour créer un fichier temporaire, il est important de créer un fichier qui n'existe pas déjà. Donc, il faut faire le test puis créer le fichier en une seule étape.

Il existe donc **mktemp** / **mkstemp** qui réalise cette opération : il est accessible au travers de la commande **mktemp** du shell ou du module **tempfile** (et en particulier la fonction **os.mkstemp**).