Signification des permissions pour les répertoires

Pour comprendre la signification des permissions pour un répertoire, il faut se souvenir de ce qu'est un répertoire.

Petit rappel : l'arborescence des répertoires sert à nommer les fichiers présents dans les systèmes de fichiers pour nous permettre d'y accéder. Le contenu d'un répertoire est une liste de paires `(nom , numéro d'inode)` appelés liens (ou liens physiques), qui permettent d'attribuer des noms à des fichiers identifiés par leur numéro d'inode.

Écrire (w) dans un répertoire

Écrire dans un répertoire correspond à modifier cette liste de liens. C'est le cas par exemple, lorsqu'on on crée un nouveau fichier ou qu'on supprime un fichier existant (dans ce cas, la ligne correspondante sera supprimée et le nombre de liens sera décrémenté dans l'inode du fichier). Ainsi, la permission d'effacer un fichier est liée aux permissions du répertoire et pas aux permissions du fichier car le contenu de celui-ci n'est pas modifié.

Exemple :

Si on crée un répertoire et qu'on supprime la permission en écriture, on ne peut pas y créer de fichier :

```
$ mkdir rep
$ cd rep
$ chmod u-w .
$ touch fic
touch: impossible de faire un touch 'fic': Permission non accordée
```

Le fichier `fic1` appartient à `root` et n'a aucune permission activée, pourtant on peut l'effacer en tant qu'user qui possède la permission en écriture sur le répertoire qui le contient :

```
$ chmod u+w .
$ touch fic1 fic2
$ chmod 0 fic1
$ sudo chown root:root fic1
$ rm fic1
rm : supprimer 'fic1' qui est protégé en écriture et est du type « fichier vide » ? y
$ ls
fic2
```

Par contre, on ne peut pas supprimer un fichier qui nous appartient lorsque le répertoire qui le contient ne le permet pas :

```
$ chmod u-w .
$ rm fic2
rm: impossible de supprimer 'fic2': Permission non accordée
```

Remarque : *certains* systèmes de fichiers permettent à l'user root de "geler" un fichier (le rendre non modifiable, non renommable, non effaçable) indépendamment des permissions du répertoire, voir `man chattr` (attributs `u`, `i`).

Lire (r) vs exécuter (x) un répertoire

Lire un répertoire correspond à accéder à l'ensemble des noms de la liste des liens (1ère colonne).

Exécuter un répertoire correspond à, connaissant le nom d'un fichier, lui associer son numéro d'inode. Souvent sur le web, on parle de "traverser" le répertoire.

Nouvel exemple :

Si on crée un fichier dans un répertoire avec les permissions en lecture et exécution, on peut le découvrir (avec la commande `ls`) et y accéder :

```
$ mkdir rep
$ echo hop > rep/plop
$ ls rep/
plop
$ cat rep/plop
hop
$ cd rep/
$ ls
plop
$ cd ..
```

Si on supprime le permission en exécution en gardant la permission en lecture, on peut voir quels fichiers il contient, mais pas accéder au contenu du fichier puisqu'on ne connaît pas son inode. On ne peut pas non plus aller dans le répertoire :

```
$ chmod u-x rep
$ cat rep/plop
cat: rep/plop: Permission non accordée
$ ls rep/
ls: impossible d'accéder à 'rep/plop': Permission non accordée
plop
$ cd rep/
bash: cd: rep/: Permission non accordée
```

Avec la permission en exécution, mais pas en lecture, on peut accéder au contenu de ses fichiers dans la mesure où connaît leur noms mais on ne peut pas les découvrir :

```
$ chmod u+x,u-r rep/
$ cat rep/plop
hop
$ cat rep/pl[TAB]               # l'autocomplétion ne marche pas
$ ls rep
ls: impossible d'ouvrir le répertoire 'rep': Permission non accordée
$ cd rep/
$ ls
ls: impossible d'ouvrir le répertoire '.': Permission non accordée
$ cat plop
hop
```

Dans un tel répertoire avec la permission en exécution mais pas la permission en lecture, on peut faire un cd dans un sous-répertoire, à condition d'en connaître le nom, c'est pour ça qu'on trouve parfois le terme de "traverser un répertoire" :

```
$ mkdir rap
```

```
$ cd rep/rap
```

Remarque : pour pouvoir accéder à un fichier nommé par un répertoire `/a/b/c/d/`, il faut que tous les répertoires qui se trouvent sur le chemin absolu du fichier soient accessibles en exécution ("traversables") ; dans notre exemple ce sont les répertoires `/`, `/a/`, `/a/b/`, `/a/b/c/` et `/a/b/c/d/`.

Par exmemple, si vous ouvrez un shell en `root` et que vous tapez :

```
# chmod -x /usr/
```

Alors, il devient impossible d'accéder aux fichiers qui se trouvent dans `/usr/` ou des sous-répertoires de `/usr/`. En particulier, la commande `ls` échoue parce que le chemin du fichier de cette commande est `/usr/bin/ls` et que `bash` ne peut pas accéder à `/usr/bin` qui se trouve dans le `PATH` :

```
$ ls
bash: ls : commande introuvable
```

Mais même si on met le chemin complet de la commande (sans se reposer sur le `PATH`), il est impossible d'accéder à la commande :

```
$ /usr/bin/ls
bash: /usr/bin/ls: Permission denied
```

On sait que `/bin` est un lien symbolique de `/usr/bin/`, mais c'est pas pour autant qu'on peut esquiver le répertoire `/usr` :

```
$ /bin/ls
bash: /bin/ls: Permission denied
```

Si on revient dans le shell exécuté par `root`, on peut remettre les bonnes permissions :

```
# chmod +x /usr/
```