NFSv4 ACLs and mode bits

Traditionally clients have used either mode bits or Windows ACLs, rarely (if ever) both.

This limits the kind of mode-bit/NFSv4 ACL interactions seen so far.

NFSv4 (hopefully!) will change this.

The goal is to move to a common permissions model based on Windows/NFSv4 ACLs.

How do we migrate unix systems to NFSv4 ACLs?

What sort of problems are we likely to encounter as we do so?

Widespread integration of Windows ACLs into unix filesystems and interfaces may present new problems not as common in previous multi-protocol situations.

Past experience with Windows ACL/mode bit sharing may not be enough.

How do applications use modes?

Do applications use modes?

Yes!

chmod calls on fieldses.org, a Debian (Sid) server/desktop/entertainment center.....

```
296077 faubackup-scatter
   832 chmod
   494 udevd
   148 dpkg
    98 sshd
    38 pine
    33 totem
    17 xmame
    17 vi
    12 dpkg-preconfigure
    11 unison
    10 ld
     6 gdm
     6 xine
     5 unison
     3 dbus-daemon
     3 postmaster
     3 screen
     3 syslogd
     3 Xorg
     2 firefox-bin
     2 objcopy
```

```
1 d
1 install
1 postmaster
1 vim
```

What do they expect from mode bits?

How do applications use mode bits?

What does the standard say?

All you need to know is:

> **After a chmod, permissions must be bounded above by the given mode.**

This condition is very easily met:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx

chmod 644

ALLOW OWNER@    rw-
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

But POSIX allows preserving more, if you want:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx

chmod 644

ALLOW OWNER@    rw-
ALLOW u:andros  r--
ALLOW u:bfields r--
ALLOW g:users   r--
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

Should you do more?

Inherited ACLs and open()

If you implement chmod by replacing the NFSv4 ACL, and you use the same algorithm on inheritance, then unix clients will **never** inherit ACLs.

Inherited ACLs represent policy that should be overridden only when explicitly requested.

Can we handle inheritance as a special case?

Based on observed chmod use by applications,

```
open("somefile", O_RDWR|O_CREAT, mode1)
... write some stuff, etc ...
chmod("somefile", mode2)
```

is extremely common.

Without ACLs, final permissions are the same as they would have been if we created the file with mode2.

With ACLs, they may be completely different.

Should we do more?

Masking

Without masking:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx

chmod 0

ALLOW EVERYONE@ ---

chmod 644

ALLOW OWNER@    rw-
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

With masking:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx

chmod 0
```

```
ALLOW EVERYONE@ ---

chmod 644

ALLOW OWNER@    rw-
ALLOW u:andros  r--
ALLOW u:bfields r--
ALLOW g:users   r--
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

How do you store the extra information?

Masking

Explicit mask in the ACL:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx

chmod 0

DENY  OWNER@    rwx
ALLOW OWNER@    rwx
DENY  u:andros  rwx
ALLOW u:andros  rwx
DENY  u:bfields rwx
ALLOW u:bfields rwx
DENY  g:users   rwx
ALLOW g:users   rwx
DENY  GROUP@    rwx
ALLOW GROUP@    rwx
DENY  EVERYONE@ rwx
ALLOW EVERYONE@ rwx

chmod 644

ALLOW OWNER@    rw-
ALLOW u:andros  r--
ALLOW u:bfields r--
ALLOW g:users   r--
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

Storing the mask separately on the server:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
```

```
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx

chmod 0

ALLOW EVERYONE@ ---

chmod 644

ALLOW OWNER@    rw-
ALLOW u:andros  r--
ALLOW u:bfields r--
ALLOW g:users   r--
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

Advantages:

- ACLs returned are as simple as in the case where there is no masking.
- We still handle the only case we care about:

```
open(...,mode1)
chmod(...,mode2)
chmod(...,mode3)
...
```

(We don't handle this case:

```
open(...,mode1)
chmod(...,mode2)
get acl
set acl
chmod(...,mode3)
```

But that's not a case we care about.)

Disadvantage:

- Operation is opaque to client

Mask attribute?

Masking

Client's view with and without mask attribute:

```
ALLOW OWNER@    rwx
ALLOW u:andros  rwx
ALLOW u:bfields rwx
ALLOW g:users   rwx
ALLOW GROUP@    rwx
ALLOW EVERYONE@ rwx
```

```
chmod 0

Client that requests acl only:  Client that requests acl and mask:

ACL:                            ACL:                    mask:

ALLOW EVERYONE@ ---             ALLOW OWNER@    rwx     000
                                ALLOW u:andros  rwx
                                ALLOW u:bfields rwx
                                ALLOW g:users   rwx
                                ALLOW GROUP@    rwx
                                ALLOW EVERYONE@ rwx

chmod 644

ALLOW OWNER@    rw-
ALLOW u:andros  r--
ALLOW u:bfields r--
ALLOW g:users   r--
ALLOW GROUP@    r--
ALLOW EVERYONE@ r--
```

Advantages:

- Operation of mask is no longer opaque to client
- Allows complete save and restore of permissions over NFSv4.
- Direct manipulation of mask may be useful in some cases.

Disadvantages:

- None. It's optional. Feel free to pretend it doesn't exist.