Network File System Version 4
Internet-Draft
Expires: February 2, 2007

A. Gruenbacher
SUSE Labs, Novell
J. Fields
CITI

August 2006

NFSv4 file_masks Attribute
draft-gruenbacher-nfsv4-file-masks-01

Status of this Memo

Copyright Notice

Abstract

Some NFSv4 servers allow a mode SETATTR to restore ACL permissions which were removed by a previous mode SETATTR. This allows servers to handle mode SETATTRs without destroying the information in ACLs. However, these temporarily masked permissions are not exposed to clients. This proposal adds an optional new file attribute, file_masks, which can be used by clients to see these temporarily masked permissions.

The operations of the NFSv4 mode and ACL attributes are unchanged, but extra information is made available to clients to aid, for example, in archiving of data without losing permissions information.

1. Problem Statement

The NFSv4 specification [1] defines both an ACL and a mode file attribute.

The NFSv4 mode attribute corresponds to the file mode on POSIX systems. POSIX requires that after the file mode is set, no process is granted more permissions than allowed by the file mode itself (definition of File Access Permissions [2]).

Therefore a server that wishes to implement the NFSv4 mode attribute in a POSIX compliant way must, after a mode SETATTR, restrict the ACL to meet the above requirement.

In the process, much or all of the information present in the original ACL can be lost. This is often undesirable. Traditional POSIX filesystems have the property that restoring a mode to a previous value will restore all permissions to the previous value, and some applications may depend on this property.

Therefore, many filesystems that support both ACLs and mode bits implement them in such a way that setting a more restrictive mode and then restoring the original mode will also restore as much of the original ACL as possible.

Filesystems do this by storing a "mask" which is independent from the rest of the ACL, and modifying only the mask on chmod(). This allows the filesystem to enforce restricted permissions without having to modify the original ACL.

A server exporting such a filesystem can return to NFSv4 clients an ACL that has the mask already applied (and hence represents the effective permissions on the file). There are advantages to also allowing the client access to the mask and the unmasked ACL:

1. The client is then able to see and manipulate all of the server's permission state beyond the effective permissions. Examples where this additional state would be useful are permission-preserving copy and backup/restore.
2. Restoring the original mode may not always completely restore the original ACL, because the ACL may grant mask flags (such as WRITE_OWNER) which go beyond the permissions covered by the mode attribute, and such permissions are usually turned off by a mode SETATTR. In this situation, the ability to explicitly set a

larger mask allows the client to restore the original ACL in its entirety if desired.

In most situations, however, an ACL representing the effective permissions on a file is still more useful. Also, clients should not be required to have knowledge of the server's masking behavior. For that reason, we must ensure that the NFSv4 ACL attribute continues to function exactly as described in RFC 3530, and we must ensure that any additional protocol is entirely optional.

2. File Masks Proposal

We propose to add an optional file_masks attribute to NFSv4. This attribute consists of a file owner, group, and other mask, each containing ACE access masks. The file masks correspond to the owner, group, and other permission bits in the mode attribute.

```
struct file_masks {
    acemask4 owner_mask;
    acemask4 group_mask;
    acemask4 other_mask;
};
```

The file_masks attribute has the following properties:

1. After an _ACL_ SETATTR:
    1. The mask flags that principals are granted are determined by _ACL_ alone.
    2. An ACL GETATTR returns _ACL_.
    3. Each of the the file masks are set to a superset of the mask flags granted to all principals with which the file mask corresponds. This guarantees that the file masks will have no effect on the permission check algorithm, as required by Paragraph 1.1.
    4. The mode attribute is set so that it reflects _ACL_.
2. After a _mode_ SETATTR:
    1. No principal shall be granted more than its corresponding file permission bits in _mode_.
    2. A mode GETATTR returns _mode_.
    3. Each of the file masks is updated based on its corresponding file permission bits in _mode_: For each file mask, if the corresponding Read, Write, and Execute permission is set, set all mask flags that are equivalent to or a subset of that permission, and clear all others. Set all mask flags that are always allowed under POSIX. (With the file masks updated based on _mode_, Paragraph 2.1 is equivalent to Paragraph 3.1.)

    4. An ACL GETATTR should return the ACL that results from applying the updated file masks to it. (This is equivalent to applying _mode _ to the ACL, which must also first convert _mode_ to the appropriate mask flags.)
3. After a _file_masks_ SETATTR:
    1. No principal shall be allowed more than its corresponding file mask in _file_masks_.
    2. A file_masks GETATTR returns _file_masks_.
    3. The file permission bits in the file mode are updated based on their corresponding file masks in _file_masks_: For each set of permissions in the file mode, set those permissions for which the corresponding file mask contains mask flags that are equivalent to or a subset of the permissions, and clear all others.
    4. An ACL GETATTR shall return the ACL that results from applying _file_masks_ to it.
4. A GETATTR for both _file_masks_ and _ACL_ shall return the file masks, together with the unmodified ACL.
5. A SETATTR of mode, ACL, and/or file_masks shall process the attributes in the order of mode, ACL, file_masks.

This proposal allows servers to implement the masking behavior described in Section 1 while avoiding the disadvantages discussed there.

3. Access Check Algorithm

When separately storing the unmodified ACL attribute and the file masks on the server, the permission check algorithm needs to take both the ACL and the file masks into account. This can be achieved by separately checking if both the ACL and the file masks permit the requested access.

The file masks can have two different meanings during access checks: they can be used to further limit the mask flags that the ACL allows, or they can limit the mask flags that the ACL allows, while at the same time defining the permissions granted to OWNER@, GROUP@, and EVERYONE@.

Both variants correspond to different ways of applying file masks to an ACL. The latter variant corresponds to having the file masks "write through" to OWNER@, GROUP@, and EVERYONE@ ACL entries, and replace their existing mask flags.

The following two sections define access check algorithms that can be used in these two cases.

3.1. Access Check Algorithm Without Write-Through

1. If the principal does not match the file owner, continue with the next paragraph. Otherwise, if the requested mask flags exceed the owner mask, deny access. Otherwise, use the NFSv4 permission check algorithm to determine access.
2. If the principal is not a member in the owning group and none of the entries in the ACL with who values other than EVERYONE@ match the principal, continue with the next paragraph. Otherwise, if the requested mask flags exceed the group mask, deny access. Otherwise, use the NFSv4 permission check algorithm to determine access.
3. If the requested mask flags exceed the other mask, deny access. Otherwise, use the NFSv4 permission check algorithm to determine access.

3.2. Access Check Algorithm With Write-Through

1. If the principal does not match the file owner, continue with the next paragraph. Otherwise, if the requested mask flags exceed the owner mask, deny access. Otherwise, allow access.
2. If the principal is not a member in the owning group, continue with the next paragraph. Otherwise, if the requested mask flags exceed the group mask, deny access. Otherwise, allow access.
3. If none of the entries in the ACL with who values other then EVERYONE@ match the principal, continue with the next paragraph. Otherwise, if the requested mask flags exceed the group mask, deny access. Otherwise, use the NFSv4 permission check algorithm to determine access.
4. If the requested mask flags exceed the other mask, deny access. Otherwise, allow access.

4. Discussion

The proposed solution meets the following goals:

- Servers and clients that do not implement the _file_masks_ attribute will be unaffected, and will not observe any changes.
- The described approach does not preclude any alternative solutions to the problems described that may exist.
- Setting the mode attribute to a permissive value will grant as many permissions in the ACL as the mode allows. Sequences of mode SETATTR are equivalent to only performing the last mode SETATTR.
- The complete permission information can be preserved when copying files, including permissions that are currently disabled.
- Clients that care can implement ACL editors that take both the ACL and the file masks into account.

The following disadvantages are known:

- The file masks will not be preserved across sequences of ACL GETATTR / ACL SETATTR.
- Independently checking if an access is granted by the ACL and by the file masks can lead to permissions that cannot be represented as an ACL, as when mode 0600 is applied to ACL "GROUP@:READ_DATA::ALLOW": in this case, only owners who are also in the owning group would be granted READ_DATA access. Granting permissions that cannot be represented as an ACL can be avoided by applying the group mask to all ACL entries with who values other than OWNER@ and EVERYONE@ during access checks.
- The proposal requires the _file_masks_ attribute to be added to the protocol, and its behavior specified, which would make a long RFC even longer.

5. References

[1] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[2] Institute of Electrical and Electronics Engineers, "Information Technology - Portable Operating System Interface (POSIX) - Base Definitions", IEEE Standard 1003.1, December 2004.

Authors' Addresses

Andreas Gruenbacher
Novell / SUSE Labs
Maxfeldstrasse 5
90409 Nuremberg

Email: agruen@suse.de

J. Bruce Fields
U. of Michigan, Center for Informaton Technology Integration (CITI)
535 West William
Ann Arbor, Michigan

Email: bfields@citi.umich.edu

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.