NFSv4 Working Group
Internet Draft
Intended Status: Standards Track
Expires: September 8, 2016

M. Naik
Nutanix
M. Eshel
IBM Almaden
March 7, 2016

File System Extended Attributes in NFSv4
draft-ietf-nfsv4-xattrs-02

Abstract

This document proposes extensions to the NFSv4 protocol which allow file extended attributes (hereinafter also referred to as xattrs) to be manipulated using NFSv4. An xattr is a file system feature that allows opaque metadata, not interpreted by the file system, to be associated with files and directories. Such support is present in many modern local file systems. New file attributes are proposed to allow clients to query the server for xattr support, and new operations to get and set xattrs on file system objects are provided.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Table of Contents

1. Introduction

Extended attributes, also called xattrs, are a means to associate opaque metadata with file system objects, typically organized in key/value pairs. They are especially useful when they add information that is not, or cannot be, present in the associated object itself. User-space applications can arbitrarily create, interrogate, and modify the key/value pairs.

Extended attributes are file system-agnostic; applications use an interface not specific to any file system to manipulate them. Applications do not need to be concerned about how the key/value pairs are stored internally within the underlying file system. All major operating systems provide various flavors of extended attributes. Many user space tools allow xattrs to be included in regular attributes that need to be preserved when objects are updated, moved or copied.

Extended attributes have previously been considered unsuitable for portable use because some aspects of their handling are not precisely defined and they are not formally documented by any standard (such as POSIX). Nevertheless, it appears that xattrs are widely deployed and their support in modern disk-based file systems is nearly universal.

There is no clear specification of how xattrs could be mapped to any existing file attributes defined in the NFSv4 protocol ([RFC7530], [RFC5661], [NFSv42]). As a result, most NFSv4 client implementations ignore application-specified xattrs. This state of affairs results in data loss if one copies, over the NFS protocol, a file with xattrs from one file system to another that also supports xattrs.

There is thus a need to provide a means by which such data loss can be avoided. This will involve exposing xattrs within the NFSv4 protocol, despite the lack of completely compatible file system implementations.

This document discusses (in Section 5) the reasons that NFSv4 named attributes as currently standardized in [RFC7530], are unsuitable for representing xattrs. Instead, it proposes a separate protocol mechanism to support xattrs. As a consequence, xattrs and named attributes will both be optional features with servers free to support either, both, or neither.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

2. Use Cases

Applications can store tracking information in extended attributes. Examples include storing metadata identifying the application that created the file, a tag to indicate when the file was last verified by a data integrity scrubber, or a tag to hold a checksum/crypto hash of the file contents along with the date of that signature. Xattrs can also be used for decorations or annotations. For example, a file downloaded from a web server can be tagged with the URL, which can be convenient if its source has to be determined in the future. Likewise, an email attachment, when saved, can be tagged with the message-id of the email, making it possible to trace the original message.

Applications may need to behave differently when handling files of varying types. For example, file managers, such as GNOME's, offer unique icons, different click behavior, and special lists of operations to perform depending on the file format. This can be achieved by looking at the file extension (Windows), or interpret the type by inspecting it (Unix MIME type). Some file managers generate this information on the fly; others generate the information once and then cache it. Those that cache the information tend to put it in a custom database. The file manager must work to keep this database in sync with the files, which can change without the file manager's knowledge. A better approach is to dispense with the custom database and store such metadata in extended attributes. This is easier to maintain, provides faster access, and is readily accessible by applications [Love].

Swift, the OpenStack distributed object store, uses xattrs to store an object's metadata along with all the data together in one file. Swift-on-File [Swift] transfers the responsibility of maintaining object durability and availability to the underlying file system. Today, this requires a native file system client to mount the volumes. Xattr support in NFS would open up the possibility of storing and consuming data from other storage systems, and facilitate the migration of data between different backend storage systems.

Baloo, the file indexing and search framework for KDE, has moved to storing metadata such as tags, ratings and comments, in file system xattrs instead of a custom database for simplicity. Starting with KDE Plasma 5.1, NFS is no longer supported due to its lack of xattr support [KDE].

3. File System Support

Extended attributes are supported by most modern file systems.

In Linux, ext3, ext4, JFS, XFS, Btrfs, among other file systems, support extended attributes. The getfattr and setfattr utilities can be used to retrieve and set xattrs. The names of the extended attributes must be prefixed by the name of the category and a dot; hence these categories are generally qualified as name spaces. Currently, four namespaces exist: user, trusted, security and system [Linux]. Recommendations on how they should be used have been published [freedesktop].

FreeBSD supports extended attributes in two universal namespaces - user and system, although individual file systems are allowed to implement additional namespaces [FreeBSD].

Solaris 9 and later allows files to have extended attributes, but implements them as "forks", logically represented as files within a hidden directory that is associated with the target file [fsattr].

In the NTFS file system, extended attributes are one of several supported "file streams" [NTFS].

Xattrs can be retrieved and set through system calls or shell commands and are generally supported by user-space tools that preserve other file attributes. For example, the "rsync" remote copy program will correctly preserve user extended attributes between Linux/ext4 and OSX/hfs by stripping off the Linux-specific "user." prefix.

4. Namespaces

Operating systems may define multiple "namespaces" in which xattrs can be set. Namespaces are more than organizational classes; the operating system may enforce different access policies and allow different capabilities depending on the namespace. Linux, for example, defines "security", "system", "trusted" and "user" namespaces, the first three being specific to Linux [freedesktop].

Implementations generally agree on the semantics of a "user" namespace, that allows applications to store arbitrary user attribute data with file system objects. Access to this namespace is controlled via the normal file system attributes. As such, getting and setting xattrs from the user namespace can be considered interoperable across platforms and vendor implementations. Attributes from other namespaces are typically platform-specific.

This document provides for namespaces supporting user-managed metadata only, thus avoiding the need to specify the semantics applicable to particular system-interpreted xattrs. The values of xattrs are considered application data just as the contents of named attributes, files, and symbolic links are. Servers have a responsibility to store whatever value the client specifies and to return it on demand. xattr keys and values MUST NOT be interpreted by the NFS clients and servers, as such behavior would lead to non-interoperable implementations. If there is a need to specify attributes that servers need to be act upon, the appropriate semantics need to be specified by adding a new attribute for the purpose as provided by [RFC7530] and [NFSv4-vers].

5. Differences from Named Attributes

[RFC7530] defines named attributes as opaque byte streams that are associated with a directory or file and referred to by a string name. Named attributes are intended to be used by client applications as a method to associate application-specific data with a regular file or directory. In that sense, xattrs are similar in concept and use to named attributes, but there are subtle differences.

File systems typically define operations to get and set individual xatrrs as being atomic, although collectively they may be independent. Xattrs generally have size limits ranging from a few bytes to several kilobytes; the maximum supported size is not universally defined and is usually restricted by the file system. Similar to ACLs, the amount of xattr data exchanged between the client and server for get/set operations can be considered to fit in a single COMPOUND request, bounded by the channel's negotiated maximum size for requests. Named attributes, on the other hand, are unbounded data streams and do not impose atomicity requirements.

Individual named attributes are analogous to files, and caching of the data for these needs to be handled just as data caching is for ordinary files following close-to-open semantics. Xattrs, on the other hand, impose caching requirements like other file attributes.

Named attributes and xattrs have different semantics and belong to disjoint namespaces. As a result, mapping one to another is, at best, a compromise.

While it should be possible to write guidance about how a client can use the named attribute mechanism to act like xattrs, such as carving out some namespace and specifying locking primitives to enforce atomicity constraints on individual get/set operations, this is problematic. A client application trying to use xattrs through named attributes with a server that supported xattrs directly would get a

lower level of service, and could fail to cooperate on a local application running on the server unless the server file system defined its own interoperability constraints. File systems that already implement xattrs and named attributes natively would need additional guidance such as reserving named attribute namespace specifically for implementation purposes.

6. XDR Description

This document contains the external data representation (XDR) [RFC4506] description of the extended attributes. The XDR description is embedded in this document in a way that makes it simple for the reader to extract into a ready-to-compile form. The reader can feed this document into the following shell script to produce the machine readable XDR description of extended attributes:

<CODE BEGINS>

```
#! /bin/sh
grep '^ *///' $* | sed 's?^ */// ??' | sed 's?^ *///$??'
```

<CODE ENDS>

That is, if the above script is stored in a file called "extract.sh", and this document is in a file called "spec.txt", then the reader can do:

```
sh extract.sh < spec.txt > xattr_prot.x
```

The effect of the script is to remove leading white space from each line, plus a sentinel sequence of "///".

The embedded XDR file header follows. Subsequent XDR descriptions, with the sentinel sequence are embedded throughout the document.

Note that the XDR code contained in this document depends on types from the proposed NFSv4.2 nfs4_prot.x file [NFSv42-dot-x]. This includes both nfs types that end with a 4, such as verifier4, count4, etc., as well as more generic types such as opaque and bool.

6.1. Code Components Licensing Notice

Both the XDR description and the scripts used for extracting the XDR description are Code Components as described in Section 4 of "Legal Provisions Relating to IETF Documents" [LEGAL]. These Code Components are licensed according to the terms of that document.

<CODE BEGINS>

```
/// /*
///  * Copyright (c) 2012 IETF Trust and the persons identified
///  * as authors of the code.  All rights reserved.
///  *
///  * Redistribution and use in source and binary forms, with
///  * or without modification, are permitted provided that the
///  * following conditions are met:
///  *
///  * o Redistributions of source code must retain the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer.
///  *
///  * o Redistributions in binary form must reproduce the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer in the documentation and/or other
///  *   materials provided with the distribution.
///  *
///  * o Neither the name of Internet Society, IETF or IETF
///  *   Trust, nor the names of specific contributors, may be
///  *   used to endorse or promote products derived from this
///  *   software without specific prior written permission.
///  *
///  *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
///  *   AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
///  *   WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
///  *   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
///  *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO
///  *   EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
///  *   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
///  *   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
///  *   NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
///  *   SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
///  *   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
///  *   LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
///  *   OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
///  *   IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
///  *   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
///  *
///  * This code was derived from RFCTBD10.
///  * Please reproduce this note if possible.
///  */

/// /*
///  * xattr_prot.x
///  */
```

```
/// /*
///  * The following include statements are for example only.
///  * The actual XDR definition files are generated separately
///  * and independently and are likely to have a different name.
///  * %#include <nfsv42.x>
///  * %#include <rpc_prot.x>
///  */
```

<CODE ENDS>

7. Protocol Extensions

This section documents extensions to the NFSv4 protocol operations to allow xattrs to be queried and modified by clients. A new attribute is added to allow clients to check if the server supports xattrs. New operations are defined to allow xattr keys and values to be queried and set. In addition, new bitmask constants are added to the ACE access mask field to validate permissions to query and modify xattrs.

These changes follow applicable guidelines for valid NFSv4 protocol extension, whether the extensions occur in a minor version (as specified in [RFC5661]) or as an extension to an existing minor version (as specified in [NFSv4-vers]).

7.1. New definitions

<CODE BEGINS>

```
/// typedef component4     xattrkey4;
/// typedef opaque         xattrvalue4<>;
```

<CODE ENDS>

Each xattr is a key/value pair. xattrkey4 is a string denoting the xattr key name, and an attrvalue4 which is a variable-length string that identifies the value of the xattr. The handling of xattrkey4 with regard to internationalization-related issues is the same as that for NFSv4 file names and named attribute names, as described in [RFC7530]. Any regular file or directory may have a set of extended attributes, each consisting of a key and associated value. The NFS client or server MUST NOT interpret the contents of xattrkey4 or xattrvalue4.

7.2. New Attribute

The following RECOMMENDED per-fs read-only attribute is proposed for use. A client can query the server to determine if xattrs are

supported by setting the xattr_support bit in the GETATTR request.

7.2.1. xattr_support

True, if the object's file system supports extended attributes.

Since xattr_support is not a REQUIRED attribute, server need not support it. However, a client may reasonably assume that a server (or file system) that does not support the xattr_support attribute does not provide xattr support and act on that basis.

Note that the protocol does not enforce any limits on the number of keys, the length of a key or the size of a value, or the total size of xattrs that are allowed for a file. The server file system MAY impose additional limits. In addition, a single xattr key or value exchanged between the client and server for get/set operations is limited by the channel's negotiated maximum size for requests and responses.

7.3. New Error Definitions

```
<CODE BEGINS>

/// /* Following lines are to be added to enum nfsstat4 */
/// /*
///  NFS4ERR_NOXATTR        = 10095 /* xattr does not exist    */
///  NFS4ERR_XATTR2BIG      = 10096 /* xattr value is too big  */
/// */

<CODE ENDS>
```

7.3.1. NFS4ERR_NOXATTR (Error Code 10095)

The specified xattr does not exist or the server is unable to retrieve it.

7.3.2. NFS4ERR_XATTR2BIG (Error Code 10096)

The size of the xattr value as part of a SETXATTR operation is bigger than that supported by the underlying file system.

7.4. New Operations

Individual xattrs generally represent separate items of metadata. For various reasons, combining them into a single attribute results in clumsy implementations with significant functional deficits. In consequence, adding a new attribute to represent the set of xattrs for an object is not an appropriate way to provide support for

xattrs.

For example, obtaining the value of a single xattr using the bitmap would require a client implementation to read all the xattrs of the file and find a match for the one requested. Similarly, replacing or deleting a single xattr while keeping the others intact would require a client to read the xattrs first, replacing the existing list with a modified list that excludes the one to be deleted, and writing out the remaining xattrs. Such a read-modify-write cycle is subject to updates being lost in the case of simultaneous updates by multiple clients. In addition, two clients might simultaneously add the same xattr key to the same file with each concluding that it did the initial creation for the common xattr key, when the semantic model implies that only one could have done so.

Applications need to perform the following operations on a given file's extended attributes [Love]:

- Given a file, return a list of all of the file's assigned extended attribute keys.
- Given a file and a key, return the corresponding value.
- Given a file, a key, and a value, assign that value to the key.
- Given a file and a key, remove that extended attribute from the file.

This section introduces four new RECOMMENDED operations, GETXATTR, SETXATTR, LISTXATTRS and REMOVEXATTR, to query, set, list and remove xattrs respectively. A server MUST support all four operations if it supports the xattr_support attribute. GETXATTR allows obtaining the value of an xattr key, SETXATTR allows creating or replacing an xattr key with a value, LISTXATTRS enumerates all the xattrs names, and REMOVEXATTR allows deleting a single xattr.

7.4.1. GETXATTR - Get an extended attribute of a file

7.4.1.1. ARGUMENTS

<CODE BEGINS>

```
/// struct GETXATTR4args {
///         /* CURRENT_FH: file */
///         xattrkey4     gxa_name;
/// };
```

<CODE ENDS>

7.4.1.2. RESULTS

<CODE BEGINS>

```
/// union GETXATTR4res switch (nfsstat4 gxr_status) {
///  case NFS4_OK:
///          xattrvalue4   gxr_value;
///  default:
///          void;
/// };
```

<CODE ENDS>

7.4.1.3. DESCRIPTION

The GETXATTR operation will obtain the value for the given extended attribute key for the file system object specified by the current filehandle.

The server will fetch the xattr value for the key that the client requests if xattrs are supported by the server for the target file system. If the server does not support xattrs on the target file system, then it MUST NOT return a value and MUST return the NFS4ERR_NOTSUPP error. The server also MUST return NFS4ERR_NOXATTR if it supports xattrs on the target but cannot obtain the requested data. If the xattr value contained in the server response is such as to cause the channel's negotiated maximum response size to be exceeded, then the server MUST return NFS4ERR_REP_TOO_BIG in gxr_status.

7.4.1.4. IMPLEMENTATION

Clients that have cached an xattr may avoid the need to do a GETXATTR by determining if the change attribute is the same as it was when the xattr was fetched. If the client does not hold a delegation for the file in question, it can do so with a GETATTR request to obtain the change attribute and comparing its value to the change attribute value fetched when the xattr value was obtained. This handling is similar to how a client would revalidate other file attributes such as ACLs.

When responding to such a GETATTR, the server will, if there is an OPEN_DELEGATE_WRITE delegation held by another client for the file in question, either obtain the actual current value of these attributes from the client holding the delegation by using the CB_GETATTR callback, or revoke the delegation. See Section 18.7.4 of [RFC5661] for details.

7.4.2. SETXATTR - Set an extended attribute of a file

7.4.2.1. ARGUMENTS

```
<CODE BEGINS>

/// enum setxattr_option4 {
///         SETXATTR4_NONE        = 0,
///         SETXATTR4_CREATE      = 1,
///         SETXATTR4_REPLACE     = 2,
/// };

/// struct SETXATTR4args {
///         /* CURRENT_FH: file */
///         setxattr_option4 sxa_option;
///         xattrkey4        sxa_key;
///         xattrvalue4      sxa_value;
/// };

<CODE ENDS>
```

7.4.2.2. RESULTS

```
<CODE BEGINS>

/// struct SETXATTR4res switch (nfsstat4 sxr_status) {
///  case NFS4_OK:
///         change_info4      sxr_info;
///  default:
///         void;
/// };

<CODE ENDS>
```

7.4.2.3. DESCRIPTION

The SETXATTR operation changes one extended attribute of a file system object. The change desired is specified by sxa_option. SETXATTR4_CREATE is used to associate the given value with the given extended attribute key for the file system object specified by the current filehandle. The server MUST return NFS4ERR_EXIST if the attribute key already exists. SETXATTR4_REPLACE is also used to set an xattr, but the server MUST return NFS4ERR_NOXATTR if the attribute key does not exist. By default (SETXATTR4_NONE), the extended attribute will be created if need be, or its value will be replaced if the attribute exists.

If the xattr key and value contained in the client request are such

that the request would exceed the channel's negotiated maximum request size, then the server MUST return NFS4ERR_REQ_TOO_BIG in sxr_status. If the server file system imposes additional limits on the size of key name or value, it MAY return NFS4ERR_XATTR2BIG.

A successful SETXATTR MUST change the file time_metadata and change attributes if the xattr is created or the value assigned to xattr changes. However, these attributes SHOULD NOT be changed if this causes no actual change in the xattr value.

On success, the server returns the change_info4 information in sxr_info. With the atomic field of the change_info4 data type, the server will indicate if the before and after change attributes were obtained atomically with respect to the SETXATTR operation. This allows the client to determine if its cached xattrs are still valid after the operation. See Section 7.6 for a discussion on xattr caching.

7.4.2.4. IMPLEMENTATION

If the object whose xattr is being changed has a file delegation that is held by a client other than the one doing the SETXATTR, the delegation(s) must be recalled, and the operation cannot proceed to actually change the xattr until each such delegation is returned or revoked. In all cases in which delegations are recalled, the server is likely to return one or more NFS4ERR_DELAY errors while the delegation(s) remains outstanding, although it might not do that if the delegations are returned quickly.

7.4.3. LISTXATTRS - List extended attributes of a file

7.4.3.1. ARGUMENTS

<CODE BEGINS>

```
/// struct LISTXATTRS4args {
///         /* CURRENT_FH: file */
///         nfs_cookie4    lxa_cookie;
///         verifier4      lxa_cookieverf;
///         count4         lxa_maxcount;
/// };
```

<CODE ENDS>

7.4.3.2. RESULTS

```
<CODE BEGINS>

/// struct LISTXATTRS4resok {
///         nfs_cookie4    lxr_cookie;
///         verifier4      lxr_cookieverf;
///         xattrkey4      lxr_names<>;
///         bool           lxr_eof;
/// };

/// union LISTXATTRS4res switch (nfsstat4 lxr_status) {
///  case NFS4_OK:
///         LISTXATTRS4resok  lxr_value;
///  default:
///         void;
/// };

<CODE ENDS>
```

7.4.3.3. DESCRIPTION

The LISTXATTRS operation retrieves a variable number of extended attribute keys from the file system object specified by the current filehandle, along with information to allow the client to request additional attribute keys in a subsequent LISTXATTRS.

The arguments contain a cookie value that represents where the LISTXATTRS should start within the list of xattrs. A value of 0 (zero) for lxa_cookie is used to start reading at the beginning of the list. For subsequent LISTXATTRS requests, the client specifies a cookie value that is provided by the server on a previous LISTXATTRS request.

The lxa_cookieverf value should be set to 0 (zero) when the lxa_cookie value is 0 (zero) (first xattr read). On subsequent requests, it should be lxr_cookieverf as returned by the server. The lxa_cookieverf must match that returned by the LISTXATTRS in which the cookie was acquired. If the server determines that the lxa_cookieverf is no longer valid for the list, the error NFS4ERR_NOT_SAME must be returned.

The lxa_maxcount value of the argument is the maximum number of bytes for the result. This maximum size represents all of the data being returned within the LISTXATTRS4resok structure and includes the XDR overhead. The server may return less data. If the server is unable to return a single xattr name within the maxcount limit, the error NFS4ERR_TOOSMALL will be returned to the client.

On successful return, the server's response will provide a list of

extended attribute keys. The "lxr_eof" flag has a value of TRUE if there are no more keys for the object.

The cookie value is only meaningful to the server and is used as a "bookmark" for the xattr key. As mentioned, this cookie is used by the client for subsequent LISTXATTRS operations so that it may continue listing keys. The cookie is similar in concept to a READDIR cookie or the READ offset but should not be interpreted as such by the client.

On success, the current filehandle retains its value.

7.4.3.4. IMPLEMENTATION

The handling of cookie/verifier is similar to that of the READDIR operation. The verifier may be used by the server to help manage cookie values that may become stale. It should be a rare occurrence that a server is unable to continue properly listing xattrs with the provided cookie/verifier pair. The server should make every effort to avoid this condition since the application at the client may not be able to properly handle this type of failure.

7.4.4. REMOVEXATTR - Remove an extended attribute of a file

7.4.4.1. ARGUMENTS

<CODE BEGINS>

```
/// struct REMOVEXATTR4args {
///         /* CURRENT_FH: file */
///         xattrkey4      rxa_name;
/// };
```

<CODE ENDS>

7.4.4.2. RESULTS

<CODE BEGINS>

```
/// struct REMOVEXATTR4res switch (nfsstat4 rxr_status) {
///  case NFS4_OK:
///         change_info4      rxr_info;
///  default:
///         void;
/// };
```

<CODE ENDS>

7.4.4.3. DESCRIPTION

The REMOVEXATTR operation deletes one extended attribute of a file system object specified by rxa_name. The server MUST return NFS4ERR_NOXATTR if the attribute key does not exist.

A successful REMOVEXATTR SHOULD change the file time_metadata and change attributes.

Similar to SETXATTR, the server communicates the value of the change attribute immediately prior to, and immediately following, a successful REMOVEXATTR operation in rxr_info. This allows the client to determine if its cached xattrs are still valid after the operation. See Section 7.6 for a discussion on xattr caching.

7.4.4.4. IMPLEMENTATION

If the object whose xattr is being removed has a file delegation that is held by a client other than the one doing the REMOVEXATTR, the delegation(s) must be recalled, and the operation cannot proceed to delete the xattr until each such delegation is returned or revoked. In all cases in which delegations are recalled, the server is likely to return one or more NFS4ERR_DELAY errors while the delegation(s) remains outstanding, although it might not do that if the delegations are returned quickly.

7.4.5. Valid Errors

This section contains a table that gives the valid error returns for each new protocol operation. The error code NFS4_OK (indicating no error) is not listed but should be understood to be returnable by all new operations. The error values for all other operations are defined in Section 13.2 of [RFC7530].

Valid Error Returns for Each New Protocol Operation

Operation	Errors
GETXATTR	NFS4ERR_ACCESS, NFS4ERR_BADXDR, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_FHEXPIRED, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PERM, NFS4ERR_REP_TOO_BIG, NFS4ERR_REP_TOO_BIG_TO_CACHE, NFS4ERR_REQ_TOO_BIG,

		NFS4ERR_RETRY_UNCACHED_REP,
		NFS4ERR_SERVERFAULT, NFS4ERR_STALE,
		NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
SETXATTR		NFS4ERR_ACCESS, NFS4ERR_BADCHAR,
		NFS4ERR_BADXDR, NFS4ERR_DEADSESSION,
		NFS4ERR_DELAY, NFS4ERR_DQUOT,
		NFS4ERR_EXIST, NFS4ERR_FHEXPIRED,
		NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED,
		NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE,
		NFS4ERR_NOSPC, NFS4ERR_OP_NOT_IN_SESSION,
		NFS4ERR_PERM, NFS4ERR_REP_TOO_BIG,
		NFS4ERR_REP_TOO_BIG_TO_CACHE,
		NFS4ERR_REQ_TOO_BIG,
		NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS,
		NFS4ERR_SERVERFAULT, NFS4ERR_STALE,
		NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
LISTXATTRS		NFS4ERR_ACCESS, NFS4ERR_DEADSESSION,
		NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_IO,
		NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG,
		NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP,
		NFS4ERR_OP_NOT_IN_SESSION,
		NFS4ERR_PERM, NFS4ERR_REP_TOO_BIG,
		NFS4ERR_REP_TOO_BIG_TO_CACHE,
		NFS4ERR_REQ_TOO_BIG,
		NFS4ERR_RETRY_UNCACHED_REP,
		NFS4ERR_SERVERFAULT, NFS4ERR_STALE,
		NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE
REMOVEXATTR		NFS4ERR_ACCESS, NFS4ERR_BADCHAR,
		NFS4ERR_BADXDR, NFS4ERR_DEADSESSION,
		NFS4ERR_DELAY, NFS4ERR_DQUOT,
		NFS4ERR_EXIST, NFS4ERR_INVAL, NFS4ERR_IO,
		NFS4ERR_LOCKED, NFS4ERR_MOVED,
		NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE,
		NFS4ERR_NOSPC, NFS4ERR_OLD_STATEID,
		NFS4ERR_OPENMODE,
		NFS4ERR_OP_NOT_IN_SESSION, NFS4ERR_PERM,
		NFS4ERR_RETRY_UNCACHED_REP, NFS4ERR_ROFS,
		NFS4ERR_SERVERFAULT, NFS4ERR_STALE,
		NFS4ERR_TOO_MANY_OPS, NFS4ERR_WRONG_TYPE

7.5. Modifications to Existing Operations

In order to provide fine-grained access control to query or modify extended attributes, additions are proposed to the set of access rights that can be checked to determine if the client is permitted to perform the xattr operation.

Note that in general, as explained in Section 18.1.4 of [RFC5661], a client cannot reliably perform an access check with only current file attributes and must verify access with the server.

This section extends the semantics of the ACCESS operation documented in Section 18.1 of [RFC5661]. Three new access permissions can be requested:

ACCESS4_XAREAD	Query a file or directory for its xattr value given a key.
ACCESS4_XAWRITE	Modify xattr keys and/or values of a file or directory.
ACCESS4_XALIST	Query a file or directory to list its xattr keys.

As with the existing access permissions, the results of ACCESS are advisory in nature, with no implication that such access will be allowed or denied in the future.

In addition, two new bitmask constants used for the access mask field are added:

ACE4_READ_XATTRS	Permission to interrogate the extended attributes of a file with GETXATTR.
ACE4_WRITE_XATTRS	Permission to change the extended attributes of a file with SETXATTR or REMOVEXATTR.
ACE4_LIST_XATTRS	Permission to list the extended attributes of a file with LISTXATTRS.

The rules for the client and server follow:

- If the client is sending ACCESS in order to determine if the user can read an xattr of the file with GETXATTR, the client SHOULD set ACCESS4_XAREAD in the request's access field.

- If the client is sending ACCESS in order to determine if the user can modify an xattr of the file with SETXATTR or REMOVEXATTR, the client SHOULD set ACCESS4_XAWRITE in the request's access field.

- If the client is sending ACCESS in order to determine if the user can list the xattr keys of the file with LISTXATTRS, the client SHOULD set ACCESS4_XALIST in the request's access field.

- If the server supports the ACE4_READ_XATTRS permission bit, it

MUST only check for it in the mode, acl, and dacl attributes when it receives an ACCESS request with ACCESS4_XAREAD set in the access field.

- If the server supports the ACE4_WRITE_XATTRS permission bit, it MUST only check for it in the mode, acl, and dacl attributes when it receives an ACCESS request with ACCESS4_XAWRITE set in the access field.

- If the server supports the ACE4_LIST_XATTRS permission bit, it MUST only check for it in the mode, acl, and dacl attributes when it receives an ACCESS request with ACCESS4_XALIST set in the access field.

Server implementations need not provide the granularity of control that is implied by this list of masks. For example, POSIX-based systems might not distinguish ACE4_XAREAD from ACE4_READ_ATTRIBUTES (or ACE4_READ_DATA); both masks would be tied to a single "stat" (or "read") permission. When such a server returns attributes to the client, it would show both ACE4_READ_ATTRIBUTES (or ACE4_READ_DATA) and ACE4_XAREAD if and only if the stat (or read) permission is enabled.

If a server receives a SETXATTR request that it cannot accurately implement, it should err in the direction of more restricted access. For example, suppose a server supports xattrs, but cannot distinguish modifying attributes from updating xattr. If a client submits an ALLOW ACE where ACE4_WRITE_ATTRIBUTES is set but ACE4_WRITE_XATTR is not (or vice versa), the server should either turn off ACE4_WRITE_ATTRIBUTES or reject the request with NFS4ERR_ATTRNOTSUPP.

7.6. Numeric Values Assigned to Protocol Extensions

This section lists the numeric values assigned new attributes and operations to implement the xattr feature. To avoid inconsistent assignments, these have been checked against the most recent protocol version [RFC5661], the current minor version [NFSv42], and all extensions currently approved as working group documents. Development of interoperable prototypes should be possible using these values, although it is possible that these values may be modified before eventual publication as a standard-track document.

<CODE BEGINS>

```
/// /*
///  * ACCESS - Check Access Rights
///  */
/// const ACCESS4_XAREAD    = 0x00000040;
/// const ACCESS4_XAWRITE   = 0x00000080;
/// const ACCESS4_XALIST    = 0x00000100;

/// /*
///  * ACE mask values
///  */
/// const ACE4_READ_XATTRS        = 0x00200000;
/// const ACE4_WRITE_XATTRS       = 0x00400000;
/// const ACE4_LIST_XATTRS        = 0x00800000;

/// /*
///  * New NFSv4 attribute
///  */
/// typedef bool            fattr4_xattr_support;

/// /*
///  * New RECOMMENDED Attribute
///  */
/// const FATTR4_XATTR_SUPPORT = 81;

/// /*
///  * New NFSv4 operations
/// */
/// /* Following lines are to be added to enum nfs_opnum4 */
/// /*
/// OP_GETXATTR               = 72,
/// OP_SETXATTR               = 73,
/// OP_LISTXATTRS             = 74,
/// OP_REMOVEXATTR            = 75,
/// */

<CODE ENDS>
```

7.7. Caching

The caching behavior for extended attributes is similar to other file attributes such as ACLs and is affected by whether OPEN delegation has been granted to a client or not.

Xattrs obtained from, or sent to, the server may be cached and clients can use them to avoid subsequent GETXATTR requests, provided that the client can ensure that the cached value has not been subsequently modified by another client. Such assurance can depend on the client holding a delegation for the file in question or the

client interrogating the change attribute to make sure that any cached value is still valid. Such caching may be read-only or write-through.

When a delegation is in effect, some operations by a second client to a delegated file will cause the server to recall the delegation through a callback. For individual operations, we describe, under IMPLEMENTATION, when such operations are required to effect a recall.

The result of local caching is that the individual xattrs maintained on clients may not be up-to-date. Changes made in one order on the server may be seen in a different order on one client and in a third order on another client. In order to limit problems that may arise due to separate operations to obtain individual xattrs and other file attributes, a client should treat xattrs just like other file attributes with respect to caching as detailed in section 10.6 of [RFC7530]. A client may validate its cached version of an xattr for a file by fetching the change attribute and assuming that if the change attribute has the same value as it did when the attributes were cached, then xattrs have not changed. If the client holds a delegation that ensures that the change attribute cannot be modified by another client, that it can dispense with actual interrogation of the change attribute.

When a client is changing xattrs of a file, it needs to determine whether there have been changes made to the file by other clients. It does this by using the change attribute as reported before and after the change operation (SETXATTR or REMOVEXATTR) in the associated change_info4 value returned for the operation. The server is able to communicate to the client whether the change_info4 data is provided atomically with respect to the change operation. If the change values are provided atomically, the client has a basis for determining, given proper care, whether other clients are modifying the file in question.

The simplest way to enable the client to make this determination is for the client to serialize all xattr changes made to a specific file. When this is done, and the server provides before and after values of the change attribute atomically, the client can simply compare the after value of the change attribute from one operation with the before value on the subsequent change operation modifying the file. When these are equal, the client is assured that no other client is modifying the file in question.

If the comparison indicates that the file was updated by another client, the xattr cache associated with the modified file is purged from the client. If the comparison indicates no modification, the xattr cache can be updated on the client to reflect the file

operation and the associated timeout can be extended. The post-operation change value needs to be saved as the basis for future change_info4 comparisons.

Xattr caching requires that the client revalidate xattr cache data by inspecting the change attribute of a file at the point when an xattr was cached. This requires that the server update the change attribute when xattrs are modified. For a client to use the change_info4 information appropriately and correctly, the server must report the pre- and post-operation change attribute values atomically. When the server is unable to report the before and after values atomically with respect to the xattr update operation, the server must indicate that fact in the change_info4 return value. When the information is not atomically reported, the client should not assume that other clients have not changed the xattrs.

The protocol does not provide support for write-back caching of xattrs. As such, all modifications to xattrs should be done by requests to the server. The server should perform such updates synchronously.

7.8. Xattrs and File Locking

Xattr operations, for the most part, function independent of operations related to file locking state. For example, xattrs can be interrogated and modified without a corresponding OPEN operation. The server does not need to check for locks that conflict with xattr access or modify operations. For example, another OPEN specified with OPEN4_SHARE_DENY_READ or OPEN4_SHARE_DENY_BOTH does not prevent access to or modification of xattrs. Note that the server MUST still verify that the client is allowed to perform the xattr operation on the basis of ACE access permissions.

However, the presence of delegations may dictate how xattr operations interact with the state-related logic. Xattrs cannot be modified when a delegation for the corresponding file is held by another client. On the other hand, xattrs can be interrogated despite the holding of a write delegation by another client since updates are write-through to the server.

7.9. pNFS Considerations

All xattr operations are sent to the metadata server, which is responsible for fetching data from and effecting necessary changes to persistent storage.

8. Security Considerations

Since xattrs are application data, security issues are exactly the same as those relating to the storing of file data and named attributes. These are all various sorts of application data and the fact that the means of reference is slightly different in each case should not be considered security-relevant. As such, the additions to the NFS protocol for supporting extended attributes do not alter the security considerations of the NFSv4.2 protocol [NFSv42].

9. IANA Considerations

The addition of xattr support to the NFSv4 protocol does not require any actions by IANA. This document limits xattr names to the user namespace, where application developers are allowed to define and use attributes as needed. Unlike named attributes, there is no namespace identifier associated with xattrs that may require registration.

10. References

10.1. Normative References

[LEGAL] IETF Trust, "Legal Provisions Relating to IETF Documents", November 2008, <http://trustee.ietf.org/docs/IETF-Trust-License-Policy.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, <http://www.rfc-editor.org/info/rfc4506>.

[RFC5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <http://www.rfc-editor.org/info/rfc5661>.

[RFC5662] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, DOI 10.17487/RFC5662, January 2010, <http://www.rfc-editor.org/info/rfc5662>.

[RFC7530] Haynes, T. and D. Noveck, "Network File System (NFS) Version 4 Protocol", RFC 7530, March 2015.

10.2. Informative References

[NFSv42] Haynes, T., Ed., "NFS Version 4 Minor Version 2", January 2016, <http://www.ietf.org/id/draft-ietf-nfsv4-minorversion2-41.txt>.

Work in progress.

[NFSv42-dot-x] Haynes, T., Ed., "NFS Version 4 Minor Version 2 Protocol External Data Representation Standard (XDR) Description", January 2016, <http://www.ietf.org/id/draft-ietf-nfsv4-minorversion2-dot-x-41.txt>.

Work in progress.

[NFSv4-vers]
D. Noveck, "NFSv4 Version Management", January 2016, <http://www.ietf.org/id/draft-ietf-nfsv4-versioning-03.txt>.

Work in progress.

[freedesktop]
"Guidelines for extended attributes", <http://www.freedesktop.org/wiki/CommonExtendedAttributes>.

[Linux] "Linux Programmer's Manual: xattr(7)", <http://man7.org/linux/man-pages/man7/xattr.7.html>.

[Love] Love, R., "Linux System Programming: Talking Directly to the Kernel and C Library", O'Reilly Media, Inc., 2007.

[FreeBSD] "FreeBSD Man Pages - extattr", <http://www.freebsd.org/cgi/man.cgi?query=extattr&sektion=9>.

[fsattr] "Oracle Man Pages - fsattr", <http://docs.oracle.com/cd/E19253-01/816-5175/6mbba7f02>.

[NTFS] "File Streams", <http://msdn.microsoft.com/en-us/library/windows/desktop/aa364404(v=vs.85).aspx>.

[Swift] "Swift-on-File", <https://github.com/stackforge/swiftonfile>.

[KDE] Handa, V., "KDE Planet", <http://vhanda.in/blog/2014/08/extended-attributes-updates/>.

Appendix A. Acknowledgements

This draft has attempted to capture the discussion on adding xattrs to the NFSv4 protocol from many participants on the IETF NFSv4 mailing list. Those who provided valuable input and comments on earlier revisions of this draft include: Tom Haynes, Christoph Hellwig, Nico Williams, Dave Noveck, Benny Halevy and Andreas Gruenbacher.

Authors' Addresses

Manoj Naik
Nutanix
1740 Technology Drive, Suite 150,
San Jose, CA 95110
Email: manoj.naik@nutanix.com

Marc Eshel
IBM Almaden
650 Harry Rd
San Jose, CA 95120
Phone: +1 408-927-1894
Email: eshel@us.ibm.com