Internet Engineering Task Force (IETF)
Request for Comments: 8276
Category: Standards Track
ISSN: 2070-1721

M. Naik
Nutanix
M. Eshel
IBM Almaden

December 2017

File System Extended Attributes in NFSv4

Abstract

This document describes an optional feature extending the NFSv4 protocol. This feature allows extended attributes (hereinafter also referred to as xattrs) to be interrogated and manipulated using NFSv4 clients. Xattrs are provided by a file system to associate opaque metadata, not interpreted by the file system, with files and directories. Such support is present in many modern local file systems. New file attributes are provided to allow clients to query the server for xattr support, with that support consisting of new operations to get and set xattrs on file system objects.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8276.

Copyright Notice

Table of Contents

1. Introduction

Extended attributes, also called xattrs, are a means to associate opaque metadata with file system objects, organized as 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 are not concerned about how the key/value pairs are stored internally within the underlying file system. All major operating systems provide facilities to access and modify extended attributes. Many user-space tools allow xattrs to be included together with regular attributes that need to be preserved when objects are updated, moved, or copied.

Extended attributes have not previously been included within the NFSv4 specification. Some issues that need to be addressed in order to be included are that, as with named attributes, some aspects of the handling of xattrs are not precisely defined and xattrs are not formally documented by any standard such as POSIX [POSIX]. Nevertheless, it appears that xattrs are widely deployed, and their support in modern disk-based file systems is nearly universal.

There is no current specification of how xattrs could be mapped to any existing file attributes defined in the NFSv4 protocol [RFC5661] [RFC7530] [RFC7862]. As a result, most NFSv4 client implementations ignore application-specified xattrs. This state of affairs results in data loss if one copies, over the NFSv4 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 [RFC5661], are unsuitable for representing xattrs. Instead, it describes 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Uses of Extended Attributes

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 GNOMEs, 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 the type can be interpreted 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].

3. Functional Gaps Due to Lack of NFSv4 Extended Attribute Support

In addition to the prospect of data loss (discussed in Section 1) that arises from use of xattrs on local file systems, application use of xattrs poses further difficulties given the current lack of xattr support within NFSv4. As a result, certain applications may not be supported by NFSv4 or may be supported in an unsatisfactory way. Some examples are discussed below.

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. At the time of writing, this requires a native file system client to mount the volumes. Xattr support in NFSv4 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 Key Distribution Exchange (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].

4. File System Support for Extended Attributes

Extended attributes are supported by most modern file systems.

Some of the file systems that support extended attributes in Linux are as follows: ext3, ext4, JFS, XFS, and Btrfs. 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 namespaces. 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].

Some file systems have facilities that are capable of storing both extended attributes and named attributes. For discussion regarding the relationship between these features, see Section 5. Solaris 9 and later provide file "forks", logically represented as files within a hidden directory that is associated with the target file [fsattr]. In the New Technology File System (NTFS), extended attributes may be stored within "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.

5. 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, which 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 support for namespaces related to 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 were a need to specify one or more attributes that servers need to act upon, the appropriate semantics would be specified by adding a new attribute for the purpose as provided for by [RFC5661] and [RFC8178].

6. Relationship with 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. Although this makes xattrs similar in concept and use to named attributes, there are important semantic differences.

File systems typically define operations to get and set individual xattrs 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 Access Control Lists (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 are opened and closed just as files are. 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, have caching requirements similar to other file attributes.

Named attributes and xattrs have different semantics and are treated by applications as belonging to disjoint namespaces. As a result, mapping from one to the other would be, at best, a compromise. Despite these differences, the underlying file system structure used to store named attributes is generally capable of storing xattrs. However, the converse is typically not the case because of the size limits applicable to xattrs.

While it might be possible to write guidance about how a client can use the named attribute mechanism to act like xattrs, such as by carving out some namespace and specifying locking primitives to enforce atomicity constraints on individual get/set operations, such an approach is sufficiently problematic; thus, it will not be attempted here. 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 a named attribute namespace specifically for implementation purposes.

7. 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 initial section 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 NFSv4.2 nfs4_prot.x file [RFC7863]. This includes both nfs types that end with a 4, such as nfs_cookie4, count4, etc., as well as more-generic types, such as opaque and bool.

To produce a compilable XDR file, the following procedure is suggested:

- Extract the file nfs4_prot.x as described in [RFC7863].
- Extract xattr_prot.x from this document as described above.
- Apply any changes required for other extensions to be included together with the xattr extension.
- Perform modifications to nfs4_prot.x as described by comments within xattr_prot.x.
- Extend the unions nfs_argop4 and nfs_resop4 to include cases for the new operations defined in this document.
- Combine the XDR files for the base NFSv4.2 protocol and all needed extensions by either concatenating the relevant XDR files or using file inclusion.

7.1. Code Components Licensing Notice

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

```
<CODE BEGINS>

/// /*
///  * Copyright (c) 2017 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 RFC 8276.
///  * Please reproduce this note if possible.
///  */

<CODE ENDS>
```

7.2. XDR for Xattr Extension

```
<CODE BEGINS>

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

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

<CODE ENDS>
```

8. 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 determine if the file system being accessed provides support for xattrs. New operations are defined to allow xattr keys and values to be queried and set. In addition, the ACCESS operation is extended by adding new mask bits to provide access information relating to xattrs.

These changes follow applicable guidelines for valid NFSv4 XDR protocol extension, as specified in [RFC8178], and obey the rules for extensions capable of being made without a change in minor version number.

8.1. 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.

8.2. New Attribute

The per-fs read-only attribute described below may be used to determine if xattrs are supported. Servers need not support this attribute, and some NFSv4.2 servers may be unaware of its existence. Before interrogating this attribute using GETATTR, a client should determine whether it is a supported attribute by interrogating the supported_attrs attribute.

8.2.1. xattr_support

xattr_support is set to True, if the object's file system supports extended attributes.

Since xattr_support is not a REQUIRED attribute, the 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 it acts on that basis.

Note that the protocol does not enforce any limits on the number of keys, the length of a key, 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.

8.3. 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>

8.3.1. NFS4ERR_NOXATTR (Error Code 10095)

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

8.3.2. NFS4ERR_XATTR2BIG (Error Code 10096)

The size of the xattr value specified as part of a SETXATTR operation, or the collective size of all xattrs of the file resulting from the SETXATTR operation, is bigger than that supported by the underlying file system.

8.4. New Operations

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.

In order to meet these requirements, this section introduces four new OPTIONAL operations: GETXATTR, SETXATTR, LISTXATTRS and REMOVEXATTR. These operations are to query, set, list, and remove xattrs, respectively. A server MUST support all four operations when they are directed to a file system that supports the xattr_support attribute and returns TRUE when it is interrogated. For file systems that either do not support the xattr_support attribute or return FALSE when the xattr_support attribute is interrogated, all of the above operations MUST NOT be supported. 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.

Note that some server implementations may not be aware of the existence of these operations, thereby a client cannot always expect that issuing one of them will either succeed or return NFS4ERR_NOTSUPP. In some cases, NFS4ERR_OP_ILLEGAL may be returned or the request may encounter an XDR decode error on the server. As a result, clients should only issue these operations after determining that support is present.

8.4.1. GETXATTR - Get an Extended Attribute of a File

8.4.1.1. ARGUMENTS

<CODE BEGINS>

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

<CODE ENDS>

8.4.1.2. RESULTS

<CODE BEGINS>

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

<CODE ENDS>

8.4.1.3. 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 or another error indicating the request was not understood. 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.

8.4.1.4. 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 obtain the change attribute with a GETATTR request and compare that change attribute's 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.

8.4.2. SETXATTR - Set an Extended Attribute of a File

8.4.2.1. ARGUMENTS

<CODE BEGINS>

```
/// enum setxattr_option4 {
///         SETXATTR4_EITHER      = 0,
///         SETXATTR4_CREATE      = 1,
///         SETXATTR4_REPLACE     = 2
/// };

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

<CODE ENDS>

8.4.2.2. RESULTS

<CODE BEGINS>

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

<CODE ENDS>

8.4.2.3. 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_EITHER), 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 the 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, it is not necessary to change these attributes if there has been no actual change in the xattr value. Avoiding attribute change in such situations is desirable as it avoids unnecessary cache invalidation.

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 8.7 for a discussion on xattr caching.

8.4.2.4. 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.

8.4.3. LISTXATTRS - List Extended Attributes of a File

8.4.3.1. ARGUMENTS

<CODE BEGINS>

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

<CODE ENDS>

8.4.3.2. RESULTS

<CODE BEGINS>

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

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

<CODE ENDS>

8.4.3.3. 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_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.

8.4.3.4. IMPLEMENTATION

The handling of a cookie is similar to that of the READDIR operation. It should be a rare occurrence that a server is unable to continue properly listing xattrs with the provided cookie. 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.

8.4.4. REMOVEXATTR - Remove an Extended Attribute of a File

8.4.4.1. ARGUMENTS

<CODE BEGINS>

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

<CODE ENDS>

8.4.4.2. RESULTS

```
<CODE BEGINS>

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

<CODE ENDS>
```

8.4.4.3. 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 MUST 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 8.7 for a discussion on xattr caching.

8.4.4.4. 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.

8.4.5. 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] and Section 11.2 of [RFC7862].

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_NOXATTR, 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_NOXATTR, 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, NFS4ERR_XATTR2BIG
LISTXATTRS	NFS4ERR_ACCESS, NFS4ERR_DEADSESSION, NFS4ERR_DELAY, NFS4ERR_INVAL, NFS4ERR_IO, NFS4ERR_MOVED, NFS4ERR_NAMETOOLONG, NFS4ERR_NOFILEHANDLE, NFS4ERR_NOTSUPP, NFS4ERR_NOXATTR, 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_NOXATTR,, 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

Valid Error Returns for Each New Protocol Operation

8.5. Modifications to Existing Operations

In order to provide fine-grained access control to query or modify extended attributes, new access rights are defined 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.

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.

8.6. Numeric Values Assigned to Protocol Extensions

This section lists the numeric values that are 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] and the current minor version [RFC7862]. Development of interoperable prototypes is possible using these values.

<CODE BEGINS>

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

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

/// /*
///  * New RECOMMENDED Attribute
///  */
/// const FATTR4_XATTR_SUPPORT = 82;

/// /*
///  * 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,
/// */

/// /*
///  * New cases for Operation arrays
///  */
/// /* Following lines are to be added to nfs_argop4 */
/// /*
/// case OP_GETXATTR:      GETXATTR4args opgetxattr;
/// case OP_SETXATTR:      SETXATTR4args opsetxattr;
/// case OP_LISTXATTRS:    LISTXATTRS4args oplistxattrs;
/// case OP_REMOVEXATTR:   REMOVEXATTR4args opremovexattr;
```

```
   /// */

   /// /* Following lines are to be added to nfs_resop4 */
   /// /*
   /// case OP_GETXATTR:      GETXATTR4res opgetxattr;
   /// case OP_SETXATTR:      SETXATTR4res opsetxattr;
   /// case OP_LISTXATTRS:    LISTXATTRS4res oplistxattrs;
   /// case OP_REMOVEXATTR:   REMOVEXATTR4res opremovexattr;
   /// */

   <CODE ENDS>
```

8.7. Caching

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

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 be based 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, 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.

An effective way to enable the client to make this determination simply is for it 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.

8.8. 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 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.

8.9. 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.

9. 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. Clients MUST NOT accord any system-interpreted semantics to xattrs, since their use is restricted to user-managed metadata only as explained in Section 5. Extended attributes are 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 protocol [RFC7530].

10. 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.

11. References

11.1. Normative References

[LEGAL] IETF Trust, "Legal Provisions Relating to IETF Documents", Version 5.0, March 2015, <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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc5661>.

[RFC7530] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", RFC 7530, DOI 10.17487/RFC7530, March 2015, <https://www.rfc-editor.org/info/rfc7530>.

[RFC7862] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <https://www.rfc-editor.org/info/rfc7862>.

[RFC7863] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 External Data Representation Standard (XDR) Description", RFC 7863, DOI 10.17487/RFC7863, November 2016, <https://www.rfc-editor.org/info/rfc7863>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <https://www.rfc-editor.org/info/rfc8178>.

11.2. Informative References

[FreeBSD] FreeBSD, "FreeBSD Manual Pages - extattr", FreeBSD System Calls Manual, January 2008, <http://www.freebsd.org/cgi/man.cgi?query=extattr&sektion=9>.

[freedesktop] freedesktop, "Guidelines for extended attributes", May 2013, <http://www.freedesktop.org/wiki/CommonExtendedAttributes>.

[fsattr] Oracle, "fsattr - extended file attributes", Man Pages Section 5: Standards, Environments, and Macros, <http://docs.oracle.com/cd/E19253-01/816-5175/6mbba7f02>.

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

[Linux] The Linux man-pages project, "Linux Programmer's Manual: xattr(7)", Linux man pages: Section 7, September 2017, <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., February 2009.

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

[POSIX] The Open Group, "System Interfaces of The Open Group Base Specifications Issue 7", IEEE Std 1003.1, 2016 Edition (HTML Version), ISBN 1937218812, September 2016, <http://pubs.opengroup.org/onlinepubs/9699919799/>.

[Swift] The OpenStack Foundation Wiki, "Swift-on-File", July 2015, <https://wiki.openstack.org/wiki/Swiftonfile>.

Acknowledgments

This document 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 draft versions of this document 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
United States of America

Email: manoj.naik@nutanix.com

Marc Eshel
IBM Almaden
650 Harry Road
San Jose, CA 95120
United States of America

Phone: +1 408-927-1894
Email: eshel@us.ibm.com