Network File System Version 4 C. Lever, Ed.
Internet-Draft Independent
Intended status: Informational 19 May 2026
Expires: 20 November 2026

The Network File System Access Control List Protocol
draft-ietf-nfsv4-nfs-acl-01

Abstract

This Informational document describes the NFS_ACL protocol. NFS_ACL is a legacy member of the Network File System family of protocols that NFS clients use to view and update Access Control Lists stored on an NFS version 2 or version 3 server.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://chucklever.github.io/i-d-nfs-acl/draft-ietf-nfsv4-nfs-acl.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-nfsv4-nfs-acl/.

Discussion of this document takes place on the Network File System Version 4 Working Group mailing list (mailto:nfsv4@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/nfsv4/. Subscribe at https://www.ietf.org/mailman/listinfo/nfsv4/.

Source for this draft and an issue tracker can be found at https://github.com/chucklever/i-d-nfs-acl.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 20 November 2026.

Table of Contents

1. Introduction

The Network File System protocol (NFS) was introduced by Sun Microsystems in the 1980s. This protocol enabled applications to access and modify files, via local POSIX system interfaces, that reside on a remote host [RFC1094].

Traditionally, permission to access files stored in NFS file systems is granted by permission bits, mimicking [POSIX]. Permission bits provide coarse-grained access control. The file owner can control only whether members of her group can read, write, or execute the file contents, or whether anyone else (without exception) has those rights.

An Access Control List, or ACL, is a mechanism that enables file owners to grant specific users fine-grained access rights to file content [IEEE].

Version 2 of NFS is described in [RFC1094], and version 3 in [RFC1813]. Neither of these protocols include a method for managing ACLs associated with files shared via the NFS protocol, even though the local file systems shared via NFS often implemented ACLs and gave local users mechanisms to read and update them.

Sun created the NFS_ACL protocol to provide that mechanism for files accessed remotely via NFS. Later, other operating systems, including Linux, implemented NFS_ACL for similar reasons.

This document describes the protocol based on the nfs_acl.x file that is publicly available in the OpenSolaris code base [OpenSolaris]. The editor has attempted to introduce no changes to the protocol as it is implemented in those operating systems and in Linux.

The document assumes readers are already familiar with the NFS version 2 or 3 protocols and at least one implementation of them.

Issues of compatibility between the protocol described in this document and NFSv4 ACLs (as described by [RFC8881]) are considered out of scope. More information on this topic is available in [I-D.ietf-nfsv4-posix-acls].

Local file systems on NFSv2 and NFSv3 servers determine the particular semantics of each Access Control List -- in other words, how the server uses each Access Control List to authorize access to file content. This document serves only as a description of the network protocol used to exchange ACLs between NFS clients and servers.

2. Conventions and Definitions

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.

As with most publications by standards bodies, this document has been published so that people may continue to create compatible implementations. However, note that, as an Informational document, this RFC does not make any compliance mandates on implementations of the protocol described herein.

3. General Concepts

3.1. A Glossary of Useful Terms

The following are a set of foundational terms used throughout this document.

application: A program that executes on a client system.

client: A computer system that utilizes compute resources provided by one or more servers.

file: A unit of data storage consisting of an ordered stream of bytes and a set of metadata attributes.

gid: A 32-bit unsigned integer that represents a group of users.

server: A computer system that provides compute resources to network peers.

uid: A 32-bit unsigned integer that represents a specific user.

user: A person logged in on a client system.

3.2. Remote Procedure Call

The Sun Remote Procedure Call (SunRPC) protocol provides a procedure-oriented interface to remote services. Each server supplies a program, which is a set of procedures. The NFS service is one such program. The combination of host address, program number, version number, and procedure number specify one remote service procedure. Servers can support multiple versions of a program that are accessed using different protocol version numbers.

The NFS and NFS_ACL protocols are both based on SunRPC. The remainder of this document assumes an NFS environment that is implemented on top of SunRPC, as it is specified in [RFC5531].

3.3. External Data Representation

The eXternal Data Representation (XDR) specification provides a standard way of representing a set of data types on a network. XDR addresses the problem of communication between network peers with different byte orders, structure alignment, and data type representation.

This document utilizes the RPC Data Description Language to specify the XDR format arguments and results to each of the RPC service procedures that an NFS_ACL server provides.

Readers can find a full guide to XDR and the RPC Data Description Language in [RFC4506].

3.3.1. Extensions to RFC 4506

The original NFS_ACL version 2 RPC language specification includes the use of the "unsigned short" type. This type is not described in [RFC4506]. However, most current implementations of the rpcgen program implement support for this type. This section provides a proper specification for "short" and "unsigned short" integer types for the RPC language, based on those implementations. This is so that the NFS_ACL version 2 RPC language specification appearing in this document accurately reflects the wire behavior of existing implementations.

The XDR wire representation of both types is a network-endian 32-bit integer, sign-extended. This maintains XDR’s consistent 4-octet alignment for all basic integer types while allowing applications to use narrower types internally.

3.3.1.1. unsigned short

The unsigned short type is zero-extended, with the high-order two octets each containing zeroes on the wire. The value range of this type is zero to 65,535, inclusive.

Example: 0xFFFF (65535) appears as 0x0000FFFF on the wire.

3.3.1.2. signed short

The short type is sign-extended, with the high-order two octets replicating the sign bit. The value range of this type is -32768 to 32767, inclusive.

When a signed short contains a positive or zero value, its high-order octets each contain 0x00. For example: 0x7FFF (32767) appears as 0x00007FFF on the wire.

When a signed short contains a negative value, its high-order octets each contain 0xFF. For example: 0xFFFF (-1) appears as 0xFFFFFFFF on the wire, and 0x8000 (-32768) appears as 0xFFFF8000 on the wire.

3.4. Authentication and Authorization

The RPC protocol includes fields in every procedure call for user authentication parameters. The specific content of the authentication parameters is determined by the type of authentication used by the server and client. A discussion of the mechanics of RPC user authentication appears in [RFC5531], in particular Sections 9 and 10.

For NFS ACLs, the user ID carried in RPC calls is used for two purposes:

* When setting an ACL via the SETACL procedure or retrieving an ACL via the GETACL procedure, the NFS_ACL service verifies that the calling user has been granted permission to perform the procedure.

* Each Access Control Entry (see below) contains an element that identifies the user to which the ACE applies. That user is represented by a 32-bit user ID. The value of the user ID in each ACE has the same meaning and mapping as the value of incoming RPC calls.

Using user ids and group ids implies that the client and server either share the same ID list or do local user and group ID mapping. Servers and clients must agree on the mapping from user to uid and from group to gid, for those sites that do not implement a consistent user ID and group ID number space. In practice, such mapping is typically performed on the server, following a static mapping scheme or a mapping established by the user from a client at mount time.

RPCSEC_GSS authentication provides stronger security through the use of cryptographic authentication. The server and client must agree on the mapping of the user's GSS principal to a local UID on the server, but the name to identity mapping is more operating system independent than the uid and gid mapping in AUTH_UNIX.

3.5. File Access Control

This section describes the abstractions that an NFS server uses to determine whether an access or modification to a file is permitted. The exact behavior of a server implementation may vary.

3.5.1. File Ownership

A file's "owner" is the designated user that is always granted permission to update that file's security attributes. As part of creating a file, the NFS server assigns the file's owner. Under normal circumstances the initial file owner is the RPC user who issued the NFS CREATE procedure. However, server security policies can mandate replacement of that user (also known as user squashing) as part of processing a CREATE procedure.

An existing file's designated owner can subsequently be changed by an NFS SETATTR procedure. After that change, the new owner is granted permission to update the file's security attributes and the old owner is no longer treated specially.

A file's "owner group" is a short list of users that have similar privileges as the file's owner, but are treated as a separate category for the purpose of permission checking.

Any user who is not a file's owner or a member of its owner group falls into the third category, known as "everyone" or "other".

3.5.1.1. Superuser Access

On most operating systems, there is a category of users known as privileged users or superusers. These users can bypass most or all access controls on files.

3.5.2. Categories of Access

In NFS versions 2 and 3, there are three rudimentary categories of access:

Read access: Read access grants permission for a user to read a file or directory.

Write access: Write access grants permission for a user to modify a file or directory.

Execute access: For a file, execute access grants permission for the user to treat the file content as executable. For a directory object, execute access grants permission for the user to perform a lookup in that directory.

3.5.3. Traditional Permission Bits

Permission bits, or mode bits, are the simplest and perhaps oldest form of access control. Each file object has a set of mode bits [POSIX].

Each of the user categories is given a set of three access type bits. Altogether there are then nine bit flags for every file object.

3.5.4. Access Control Lists

An Access Control Entry, or ACE, represents a set of access categories and a specific user or group. An Access Control List is a list of ACEs.

Mode bits, as explained in the previous section, are essentially an ACL that always contains exactly three ACEs: one for the file’s owner, one for the file’s owner group, and one for everyone else.

3.5.4.1. Interpreting Access Control Lists

NFS clients do not perform access checks based on their interpretation of an ACL read from the server. NFS servers are solely responsible for authorizing and restricting access to file content via the NFS protocol.

An NFS Access Control List is a list of three or more Access Control Entries (ACEs) associated with one file system object. Each Access Control Entry in this list specifies a user and a set of access types granted to that user.

Only ACEs that match the requester are considered. Each ACE is processed until all of the bits of the requester’s access have been ALLOWED. Once a bit has been ALLOWED, that bit is no longer considered in the processing of subsequent ACEs in the list.

When the ACL has been fully processed, if there are bits in the requester’s mask that have not been ALLOWED, access of that type is denied.

Note that an ACL might not be the sole determiner of access. For example:

* In the case of a file system exported as read-only, the server may deny write access even though an object's ACL grants it.

* Server implementations can grant some limited permission to update an ACL in order to prevent a situation from rising in which there is no valid way to ever modify the ACL.

* All servers will allow a user the ability to read the data of the file when only the execute permission is granted (i.e., if the ACL denies the user the NA_READ access and allows the user NA_EXEC, the server will allow the user to read the data of the file).

* Some server implementations have the notion of owner-override, in which the owner of the object is allowed to override accesses that are denied by the ACL. This can be helpful, for example, to allow users continued access to open files on which the permissions have changed.

* Some server implementations have the notion of a "superuser" that has privileges beyond an ordinary user. The superuser may be able to read or write data or metadata in ways that would otherwise not be permitted by the object's ACL.

NFS clients can use either the NFS_ACL version 2 ACCESS procedure or the NFS version 3 ACCESS procedure to ask the server to perform an access check based on the requesting user and the ACL present on a file system object. Clients are also free to simply try an operation to see what works, then recover it the server denies access.

3.5.4.2. ACLs in Operation

The SETACL procedure sets two types of Access Control Lists:

Access: An NFS access ACL specifies the access permission for a file object. Access Control Entries in an ACL's "aclent" field comprise the object's access ACL.

Default: An NFS default ACL specifies the default ACL that is set on objects that are children of a directory. Access Control Entries in an ACL's "dfaclent" comprise an object's default ACL. The default ACL does not affect access to the object on which it is set.

Each NFS ACL must have one ACE for each of NA_USER_OBJ, NA_GROUP_OBJ, and NA_OTHER_OBJ. An NFS ACL that consists only of these three ACEs is referred to as a minimal NFS ACL.

An NFS ACL may zero or more NA_USER and/or NA_GROUP ACEs.

When a client presents a SETACL operation that a server finds is invalid or it cannot process, the server responds with ACL2ERR_IO or ACL3ERR_INVAL, depending on the version of NFS_ACL that is in use. ACLs that are not valid include:

* The presented ACL does not contain one ACE for each of NA_USER_OBJ, NA_GROUP_OBJ, and NA_OTHER_OBJ

* The presented ACL is a default ACL but the target object is not a directory

* The presented ACL contains too many ACEs

* The presented ACL's type field contains more than one set bit

| cel: What does the NA_ACL_DEFAULT bit do?

| rthurlow: In the Solaris acl(2)/facl(2) system calls, the
| equivalently-defined ACL_DEFAULT is used after aclent sorting
| to note where regular entries end and where default entries
| start in a single list. So perhaps it would be normal for the
| dfaclent entries to all be so marked.

The "id" field in an Access Control Entry is interpreted as follows:

* For an ACE that specifies an NA_USER_OBJ, NA_USER, NA_GROUP, and NA_GROUP_OBJ, the "id" field contains a UID or GID value that identifies the user on the server whose access permission is being set.

* For an ACE that specifies other types of permission, the "id" field is ignored.

3.5.4.3. Relationship Between ACLs and Other File Attributes

When an ACL is present on a file, the ACL controls the requesting user's access to the file. Typically the NFS server ignores the file's mode bits.

Depending on the behavior of the local file system implementation, changing the file's ACL via the SETACL procedure may alter the file's mode bits, and changing the mode bits via the SETATTR procedure may alter the content of the ACL in any way. NFS clients should refresh cached ACLs or file modes after one of these operations.

When an ACL is present on a file, changing the file's owner (say, via the SETATTR operation) may alter the server's interpretation of any ACE that targets NA_USER_OBJ.

When an ACL is present on a file, changing the file's group (say, via the SETATTR operation) may alter the server's interpretation of any ACE that targets NA_GROUP_OBJ.

If an NFS client observes that a file's ctime attribute has changed, it should assume that any ACLs that are present might have been modified.

3.5.4.4. ACL Inheritance

> | Section needs to explain how default ACLs work and what impact
> | they may have on the mode bits of the new file.

A client uses one of the NFS CREATE, MKDIR, or MKNOD procedures to request instantiation of a new file object. The server uses the default ACL from the parent directory as the initial access ACL on the new object.

3.5.4.5. Historical References

The section entitled "The POSIX 1003.1e/1003.2c Working Group" in [Gruenbacher] details the history of POSIX standards efforts with regard to file access control. The editor recommends that readers familiarize themselves with the extent to which POSIX specifies the content and behavior of ACLs.

4. Protocol Elements Common to Both Versions

4.1. RPC Authentication

The NFS_ACL service uses AUTH_NONE in the NULL procedure. All RPC authentication flavors may be used for other procedures.

4.2. Constants

These are the RPC constants needed to call the NFS Version 3 service. They are given in decimal.

100227 The RPC program number for the NFS_ACL protocol

Only versions 2 and 3 of this RPC program are valid.

4.3. Transport address

The NFS_ACL protocol can operate over the TCP, UDP, and RDMA transport protocols. For TCP and UDP, it uses port 2049, and for RDMA, it uses 20049. In both cases, this is the same as the base NFS protocol.

4.4. Sizes

NFS_ACL_MAX_ENTRIES 1024

The maximum number of Access Control Entries allowed in one Access Control List array.

4.5. Basic Data Types

The following XDR definitions are basic scalar types that are used in other structures.

```
typedef unsigned int uid;

typedef unsigned short o_mode;
```

4.6. Structured Data types

The following XDR definitions are common structured data types that are used in all versions of the NFS_ACL protocol.

4.6.1. aclent

This structure represents a single entry in an Access Control List.

```
struct aclent {
    int type;
    uid id;
    o_mode perm;
};
```

The "type" element in an Access Control Entry is a bit mask. The bit field values in this mask are defined as follows:

```
const NA_USER_OBJ = 0x1;        /* object owner */
const NA_USER = 0x2;            /* additional users */
const NA_GROUP_OBJ = 0x4;       /* owning group of the object */
const NA_GROUP = 0x8;           /* additional groups */
const NA_CLASS_OBJ = 0x10;      /* file group class and mask entry */
const NA_OTHER_OBJ = 0x20;      /* other entry for the object */
const NA_ACL_DEFAULT = 0x1000;  /* default flag */
```

The "perm" element in an Access Control Entry is also a bit mask. The bit field values in this mask are defined as follows:

```
const NA_READ = 0x4;            /* read permission */
const NA_WRITE = 0x2;           /* write permission */
const NA_EXEC = 0x1;            /* exec permission */
```

4.6.2. secattr

The secattr structure represents, on the wire, the full Access Control List for one file system object. This list contains an array of Access Control Entries that apply to the object, plus an array of default Access Control Entries that are inherited by the object’s children.

```
struct secattr {
    unsigned int mask;
    int aclcnt;
    aclent aclent<NFS_ACL_MAX_ENTRIES>;
    int dfaclcnt;
    aclent dfaclent<NFS_ACL_MAX_ENTRIES>;
};
```

The "mask" element of the secattr structure is a bit mask. The bit field values in this mask are defined as follows:

```
const NA_ACL = 0x1;         /* aclent contains a valid list */
const NA_ACLCNT = 0x2;      /* number of entries in the aclent list */
const NA_DFACL = 0x4;       /* dfaclent contains a valid list */
const NA_DFACLCNT = 0x8;    /* number of entries in the dfaclent list */
```

These bit field values are also used in the "mask" element of the GETACL2args and GETACL3args structures.

4.6.3. Interoperability Considerations

Interoperability between NFS peers that do not implement the NFS_ACL protocol is what we already have today. Interoperability between peers that both implement the NFS_ACL protocol is described in the rest of this document.

The following subsections briefly discuss three new interoperability scenarios.

4.6.3.1. Client Implements NFS_ACL, Server Does Not

Typically an NFS server that implements the NFS_ACL program will advertise the presence of NFS_ACL via an rpcbind registration. An NFS client that implements NFS_ACL should perform an rpcbind query before attempting any NFS_ACL procedure [RFC1833].

If the client sends any NFS_ACL procedure without sending an rpcbind query first, and the server does not implement the NFS_ACL program, the server responds with an RPC access_stat of PROG_UNAVAIL.

4.6.3.2. Server Implements NFS_ACL, Client Does Not

An NFS server that implements advanced access control can deny requests made by a client by responding with NFS2ERR_ACCESS or NFS3ERR_ACCESS status codes, and an NFS client has no visibility as to why the denial occurred. Neither can that client send operations to update the access control on file objects.

This is a quality of implementation issue for the client.

4.6.3.3. Client Implements, Exported File System Does Not

An NFS server that implements the NFS_ACL protocol might share both file systems that implement ACLs and file systems that do not. In this case, NFS clients detect the presence of an NFS_ACL service on the NFS server.

For file objects that do not implement ACL support:

* The server responds to a GETACL procedure by returning a manufactured minimal ACL (i.e., only three ACEs) that reflects the current mode bits of the object.

* The server responds to a SETACL version 3 procedure by returning ACL3ERR_NOTSUPP.

* The server responds to a SETACL version 2 procedure by returning ACL2ERR_IO.

The Linux implementation of the NFS_ACL protocol deviates from the protocol specified in the current document by returning NFS3ERR_NOTSUPP in response to a SETACL version 2 procedure on a file system that does not support ACLs.

5. NFS_ACL Version 2

Version 2 of the NFS_ACL protocol is used in conjunction only with version 2 of the NFS protocol.

5.1. Data types inherited from NFS version 2

5.1.1. ftype

The enumeration "ftype" gives the type of an NFS version 2 file. This definition comes from Section 2.3.2 of [RFC1094]:

```
enum ftype {
    NFNON = 0,
    NFREG = 1,
    NFDIR = 2,
    NFBLK = 3,
    NFCHR = 4,
    NFLNK = 5
};
```

5.1.2. fhandle

NFS version 2 uses a fixed-size file handle. The following definition comes from Section 2.3.3 of [RFC1094]:

```
const FHSIZE = 32;

typedef opaque fhandle[FHSIZE];
```

5.1.3. timeval

NFS version 2's "timeval" structure represents the number of seconds and microseconds since midnight January 1, 1970, Greenwich Mean Time. This definition comes from Section 2.3.4 of [RFC1094]:

```
struct timeval {
    unsigned int seconds;
    unsigned int useconds;
};
```

5.1.4. nfsfattr

This document refers to NFS version 2's file attribute structure as "nfsfattr". This is the same as the fattr structure described in Section 2.3.5 of [RFC1094]:

```
struct fattr {
    ftype        type;
    unsigned int mode;
    unsigned int nlink;
    unsigned int uid;
    unsigned int gid;
    unsigned int size;
    unsigned int blocksize;
    unsigned int rdev;
    unsigned int blocks;
    unsigned int fsid;
    unsigned int fileid;
    timeval      atime;
    timeval      mtime;
    timeval      ctime;
};
```

5.1.5. Defined Error Numbers

Section 2.3.1 of [RFC1094] describes an enumerated type called "stat" which provides a status code for NFS version 2 results. A matching type called "aclstat2" is defined in this document for the similar purpose of returning NFS_ACL version 2 procedure status codes. The numeric values of these two types match up, though aclstat2 omits some codes that are not relevant to the NFS_ACL protocol.

```
enum aclstat2 {
    ACL2_OK = 0,
    ACL2ERR_PERM = 1,
    ACL2ERR_NOENT = 2,
    ACL2ERR_IO = 5,
    ACL2ERR_ACCES = 13,
    ACL2ERR_NOSPC = 28,
    ACL2ERR_ROFS = 30,
    ACL2ERR_DQUOT = 69,
    ACL2ERR_STALE = 70
};
```

These status codes carry the following meanings:

ACL2ERR_PERM Not owner. The caller does not have correct ownership to perform the requested operation.

ACL2ERR_NOENT No such file or directory. The file or directory name specified does not exist.

ACL2ERR_IO Some sort of hard error occurred when the operation was in progress. This could be a disk error, for example.

ACL2ERR_ACCES Permission denied. The caller does not have the correct permission to perform the requested operation.

ACL2ERR_NOSPC No space left on device. The operation caused the server's file system to reach its limit.

ACL2ERR_ROFS Read-only file system. Write attempted on a read-only file system.

ACL2ERR_DQUOT Disk quota exceeded. The client's disk quota on the server has been exceeded.

ACL2ERR_STALE The "fhandle" given in the arguments was invalid. That is, the file referred to by that file handle no longer exists, or access to it has been revoked.

5.2. Server Procedures

5.2.1. Procedure 0: NULL - No Operation

5.2.1.1. ARGUMENTS

void;

5.2.1.2. RESULTS

void;

5.2.1.3. DESCRIPTION

This is the usual NULL procedure with a void argument and void result.

5.2.1.4. IMPLEMENTATION

It is important that this procedure do no work at all so that clients can use it to measure the overhead of processing a service request. By convention, the NULL procedure should never require any authentication. A server implementation may choose to ignore this convention, if responding to the NULL procedure call acknowledges the existence of a resource to an unauthenticated client.

5.2.1.5. ERRORS

Since the NULL procedure returns no result, it can not return an NFS_ACL error status code. However, some server implementations may return RPC-level errors based on security or authentication policy settings.

5.2.2. Procedure 1: GETACL - Retrieve an Access Control List

5.2.2.1. ARGUMENTS

```
struct GETACL2args {
    fhandle fh;
    unsigned int mask;
};
```

5.2.2.2. RESULTS

```
struct GETACL2resok {
    struct nfsfattr attr;
    secattr acl;
};

union GETACL2res switch (enum nfsstat status) {
case ACL2_OK:
    GETACL2resok resok;
default:
    void;
};
```

5.2.2.3. DESCRIPTION

The GETACL procedure retrieves Access Control List information associated with the file system object specified by the GETACL2args.fh field. The client obtains this file handle using one of the NFS version 2 LOOKUP, CREATE, MKDIR, or SYMLINK procedures, or the MOUNT service, as described in [RFC1094].

The GETACL2args.mask field specifies which information is to be returned in the response:

* If the NA_ACL bit is set, the server fills in the object's access ACL.

* If the NA_ACLCNT bit is set, the server fills in the number of ACEs that are in the object's access ACL.

* If the NA_DFACL bit is set, the server fills in the object's default ACL.

* if the NA_DFACLCNT bit is set, the server fills in the number of ACEs that are in the object's default ACL.

If the GETACL procedure is successful, the server sets the GETACL2res.status field to ACL2_OK. It fills in the GETACL2resok.attr field with the file object's current file attributes, as detailed in [RFC1094]. Lastly, it fills in the GETACL2res.acl field with two counted arrays of Access Control Entries (ACEs).

Otherwise, GETACL2res.status contains an error status on failure and no other results are returned.

5.2.2.4. IMPLEMENTATION

When GETACL2args.fh represents a file object that does not currently have an ACL associated with it or does not implement support for ACLs, the server responds by returning a manufactured minimal NFS ACL that reflects the current owner, group, and mode bits of the object (see Section 3.5.4.2).

5.2.2.5. ERRORS

* ACL2ERR_IO

* ACL2ERR_STALE

5.2.3. Procedure 2: SETACL - Set or replace an Access Control List

5.2.3.1. ARGUMENTS

```
struct SETACL2args {
    fhandle fh;
    secattr acl;
};
```

5.2.3.2. RESULTS

```
struct SETACL2resok {
    struct nfsfattr attr;
};

union SETACL2res switch (enum nfsstat status) {
case ACL2_OK:
    SETACL2resok resok;
default:
    void;
};
```

5.2.3.3. DESCRIPTION

The SETACL procedure replaces the Access Control Lists associated with the file system object specified by the SETACL2args.fh field with the ACLs specified by the SETACL2args.acl field. The client obtains the file handle using one of the NFS version 2 LOOKUP, CREATE, MKDIR, SYMLINK procedures, or the MOUNT service, as described in [RFC1094].

To remove extended access control from a file object, a client uses SETACL to replace the object's ACL with a minimal NFS ACL (see Section 3.5.4.2).

If the SETACL procedure is successful, the server sets the SETACL2res.status field to ACL2_OK and fills in the SETACL2resok.attr field with the file object's new file attributes, as detailed in [RFC1094].

Otherwise, SETACL2res.status contains an error status on failure and no other results are returned.

5.2.3.4. IMPLEMENTATION

On success, the server does not send the reply until the ACL change is durable locally.

Changing a file object's ACL changes the object's mtime. The mtime change is reflected in the attributes returned in the SETACL response.

A high-quality server implementation ensures that a GETACL procedure running concurrently with a SETACL procedure does not return partially updated (torn) ACL contents. However, a failed SETACL may partially change a file's ACLs.

When SETACL2args.fh represents a file object that does not implement support for ACLs, or the new ACL does not contain at least the minimal set of ACEs (as described in Section 3.5.4.2), the server responds by setting SETACL2res.status to ACL2ERR_IO.

5.2.3.5. ERRORS

* ACL2ERR_ROFS
* ACL2ERR_PERM
* ACL2ERR_IO

* ACL2ERR_ACCES

* ACL2ERR_NOSPC

* ACL2ERR_DQUOT

* ACL2ERR_STALE

5.2.4. Procedure 3: GETATTR - Get file attributes

5.2.4.1. ARGUMENTS

```
struct GETATTR2args {
    fhandle fh;
};
```

5.2.4.2. RESULTS

```
struct GETATTR2resok {
    struct nfsfattr attr;
};

union GETATTR2res switch (enum nfsstat status) {
case ACL2_OK:
    GETATTR2resok resok;
default:
    void;
};
```

5.2.4.3. DESCRIPTION

The GETATTR procedure retrieves the current file attributes associated with the file system object specified by the GETATTR2args.fh field. The client obtains this file handle using one of the NFS version 2 LOOKUP, CREATE, MKDIR, SYMLINK procedures, or the MOUNT service, as described in [RFC1094].

If the GETATTR procedure is successful, the server sets the GETATTR2res.status field to ACL2_OK, and fills in the GETATTR2resok.attr field with the file object's current file attributes, as detailed in [RFC1094].

Otherwise, GETATTR2res.status contains an error status on failure and no other results are returned.

5.2.4.4. IMPLEMENTATION

Refer to Section 2.3.5 of [RFC1094] for details about the content of the returned file attributes.

5.2.4.5. ERRORS

* ACL2ERR_IO

* ACL2ERR_STALE

5.2.5. Procedure 4: ACCESS - Check access permission

5.2.5.1. ARGUMENTS

```
struct ACCESS2args {
    fhandle fh;
    uint32 access;
};
```

5.2.5.2. RESULTS

```
const ACCESS2_READ = 0x1;       /* read data or readdir a directory */
const ACCESS2_LOOKUP = 0x2;     /* lookup a name in a directory */
const ACCESS2_MODIFY = 0x4;     /* rewrite existing file data or */
                                /* modify existing directory entries */
const ACCESS2_EXTEND = 0x8;     /* write new data or add directory entries */
const ACCESS2_DELETE = 0x10;    /* delete existing directory entry */
const ACCESS2_EXECUTE = 0x20;   /* execute file (no meaning for a directory) */

struct ACCESS2resok {
    struct nfsfattr attr;
    uint32 access;
};

union ACCESS2res switch (enum nfsstat status) {
case ACL2_OK:
    ACCESS2resok resok;
default:
    void;
};
```

5.2.5.3. DESCRIPTION

The ACCESS procedure determines the access rights that a user, as identified by the RPC credentials in the request, has with respect to the file handle specified by the ACCESS2args.fh field. The client obtains this file handle using one of the NFS version 2 LOOKUP, CREATE, MKDIR, SYMLINK procedures, or the MOUNT service, as described in [RFC1094]. The client encodes the set of permissions that are to be checked in the ACCESS2args.access field.

The following access permissions may be requested:

ACCESS2_READ Read data from file or read a directory.

ACCESS2_LOOKUP Look up a name in a directory (no meaning for non-directory objects).

ACCESS2_MODIFY Rewrite existing file data or modify existing directory entries.

ACCESS2_EXTEND Write new data or add directory entries.

ACCESS2_DELETE Delete an existing directory entry (no meaning for non-directory objects).

ACCESS2_EXECUTE Execute file (no meaning for a directory).

If the ACCESS procedure is successful, the server sets the ACCESS2res.status field to ACL2_OK. It fills in the ACCESS2resok.attr field with the file object’s current file attributes, as detailed in [RFC1094]. Lastly, it encodes the set of permissions that the requesting user is granted in the ACCESS2resok.access field.

5.2.5.4. IMPLEMENTATION

In the NFS version 2 protocol, the only reliable way to determine whether an operation is allowed is to try it and see if it succeeded or failed. Using the ACCESS procedure in the NFS_ACL version 2 protocol, a client can ask the server to indicate whether or not one or more classes of operations are permitted.

In general, it is not sufficient for a client to attempt to deduce access permissions by inspecting the uid, gid, and mode fields in the file attributes, since the server may perform uid or gid mapping or enforce additional access control restrictions. It is also possible that the NFS version 2 protocol server may not be in the same ID space as the NFS version 2 protocol client. In these cases, the NFS version 2 protocol client can not reliably perform an access check with only current file attributes.

The information returned by the server in response to an ACCESS call is advisory only. It was correct at the exact time that the server performed the checks, but not necessarily afterwards. The server can revoke access permission to a file object at any time.

The NFS_ACL version 2 protocol client should use the effective credentials of the user to build the authentication information in the ACCESS request used to determine access rights. It is the effective user and group credentials that are used in subsequent read and write operations.

Many implementations do not directly support the ACCESS2_DELETE permission. Operating systems like UNIX may ignore the ACCESS2_DELETE bit if set on an access request on a non-directory object. In these systems, delete permission on a file is determined by the access permissions on the directory in which the file resides, instead of being determined by the permissions of the file itself. Thus, the bit mask returned for such a request will have the ACCESS2_DELETE bit set to 0, indicating that the client does not have this permission.

The server should return a status of ACL2_OK if no errors occurred that prevented the server from making the required access checks.

5.2.5.5. ERRORS

* ACL2ERR_IO

* ACL2ERR_STALE

5.2.6. Procedure 5: GETXATTRDIR - Get named attribute directory

5.2.6.1. ARGUMENTS

```
struct GETXATTRDIR2args {
    fhandle fh;
    bool create;
};
```

5.2.6.2. RESULTS

```
struct GETXATTRDIR2resok {
    fhandle fh;
    struct nfsfattr attr;
};

union GETXATTRDIR2res switch (enum nfsstat status) {
case ACL2_OK:
    GETXATTRDIR2resok resok;
default:
    void;
};
```

5.2.6.3. DESCRIPTION

Section 5.3 of [RFC8881] defines a set of generic file attributes known as "named attributes". The GETXATTRDIR procedure extends this facility into the NFSv2 protocol.

The GETXATTRDIR procedure obtains the file handle of the named attribute directory associated with the file handle in the GETXATTRDIR2args.fh field. This directory contains only objects of type NFREG.

If the GETXATTRDIR procedure is successful, the server sets the GETXATTRDIR2res.status field to ACL2_OK. It fills in the GETXATTRDIR2resok.fh field with a file handle that the client may use to look up the target file's named attributes. It fills in the GETXATTRDIR2resok.attr field with the name attribute directory's current file attributes, as detailed in [RFC1094].

Using the file handle returned in GETXATTRDIR2resok.fh, a client can utilize the READDIR and LOOKUP procedures to obtain file handles for the named attributes associated with the target file system object.

If the target file object does not currently have a named attribute directory associated with it and the GETXATTRDIR2args.create boolean field is set to false, the server returns ACL2ERR_NOENT. If the target file object does not currently have a named attribute directory associated with it and the GETXATTRDIR2args.create boolean field is set to true, the server attempts to create the named attribute directory before returning a result. If the target file currently has a named attribute directory associated with it and the GETXATTRDIR2args.create boolean is set to true, the server returns the file handle of that named attribute directory.

If the RPC user does not have read access to the target file, or if the GETXATTRDIR operation is to create a named attribute directory and the RPC user does not have permission to do so, the server returns ACL2ERR_ACCES in the GETXATTRDIR2.status field.

If the target file handle designates an object not of type NFREG or NFDIR, the server returns the value ACL2ERR_IO in the GETXATTRDIR2.status field. Neither named attributes nor named attribute directories have their own named attributes.

Note: This operation is equivalent to the NFSv4 OPENATTR operation as specified in Section 16.17 of [RFC7530] and Section 18.17 of [RFC8881].

5.2.6.4. IMPLEMENTATION

Server implementers are free to choose not to implement this procedure. In this case, the server returns the RPC-level error PROC_UNAVAIL.

If the server implementation does implement the GETXATTRDIR procedure but the shared file system containing the file object specified by the file handle in the GETXATTRDIR2args.fh field does not support named attributes, the server returns ACL2ERR_IO in the GETXATTRDIR2.status field.

5.2.6.5. ERRORS

* ACL2ERR_PERM
* ACL2ERR_NOENT
* ACL2ERR_IO
* ACL2ERR_ACCES
* ACL2ERR_NOSPC
* ACL2ERR_ROFS
* ACL2ERR_STALE

6. NFS_ACL Version 3

Version 3 of the NFS_ACL protocol is used in conjunction only with version 3 of the NFS protocol.

6.1. Data types inherited from NFS version 3

6.1.1. Scalar Data types

These are defined in Section 2.5 of [RFC1813].

```
typedef unsigned hyper uint64;

typedef unsigned long uint32;

typedef uint64 fileid3;

typedef uint32 uid3;

typedef uint32 gid3;

typedef uint64 size3;

typedef uint32 mode3;
```

6.1.2. ftype3

The enumeration "ftype3" represents the type of a file object. This definition is further explained in Section 2.6 of [RFC1813].

```
enum ftype3 {
    NF3REG    = 1,
    NF3DIR    = 2,
    NF3BLK    = 3,
    NF3CHR    = 4,
    NF3LNK    = 5,
    NF3SOCK   = 6,
    NF3FIFO   = 7
};
```

6.1.3. specdata3

```
struct specdata3 {
    uint32     specdata1;
    uint32     specdata2;
};
```

The interpretation of the two words depends on the type of file system object. For a block special (NF3BLK) or character special (NF3CHR) file, specdata1 and specdata2 are the major and minor device numbers, respectively. For all other file types, these two elements should either be set to 0 or the values should be agreed upon by the client and server.

Further detail is available in Section 2.6 of [RFC1813].

6.1.4. nfs_fh3

The nfs_fh3 data type is a variable-length opaque object returned by the NFS version 3 LOOKUP, CREATE, SYMLINK, MKNOD, LINK, or READDIRPLUS procedures. A client uses this handle during subsequent NFS operations to reference the file. This definition comes from Section 2.6 of [RFC1813].

NFS3_FHSIZE 64

The maximum size in bytes of the opaque file handle.

```
struct nfs_fh3 {
    opaque       data<NFS3_FHSIZE>;
};
```

To the client, a file handle is opaque. The client stores file handles for use in a later request and can compare two file handles from the same server for equality by doing a byte-by-byte comparison, but cannot otherwise interpret the contents of file handles. Further, if two file handles from the same server are equal, they must refer to the same file, but if they are not equal, no conclusions can be drawn.

Servers may revoke access provided by a file handle at any time. If the file handle passed in a call refers to a file system object that no longer exists on the server or access for that file handle has been revoked, the error, ACL3ERR_STALE, is returned.

6.1.5. nfstime3

NFS version 3's "nfstime3" structure represents the number of seconds and nanoseconds since midnight January 1, 1970 Greenwich Mean Time. Further details are in Section 2.6 of [RFC1813].

```
struct nfstime3 {
    uint32   seconds;
    uint32   nseconds;
};
```

6.1.6. nfsfattr3

This document refers to NFS version 3's file attribute structure as "nfsfattr3". This is the same as the fattr3 structure described in Section 2.6 of [RFC1813]. A definition of the bit fields in the "mode" element, which relate to traditional file system access permissions, can also be found there.

```
struct fattr3 {
    ftype3     type;
    mode3      mode;
    uint32     nlink;
    uid3       uid;
    gid3       gid;
    size3      size;
    size3      used;
    specdata3  rdev;
    uint64     fsid;
    fileid3    fileid;
    nfstime3   atime;
    nfstime3   mtime;
    nfstime3   ctime;
};
```

6.1.7. post_op_attr

The NFS version 3 "post_op_attr" data type returns file attributes that are not directly involved in the requested procedure. See Section 2.6 of [RFC1813] for more information.

```
union post_op_attr switch (bool attributes_follow) {
case TRUE:
    fattr3   attributes;
case FALSE:
    void;
};
```

The format of this data type appears to make returning file attributes optional. However, server implementers are strongly encouraged to make a best effort to return attributes whenever possible, even when returning an error.

6.2. Error Values

Section 2.5 of [RFC1813] describes an enumerated type called "nfsstat3" which provides a status code for NFS version 3 procedure results. A matching type called "aclstat3" is defined in this document for the similar purpose of returning NFS_ACL version 3 procedure status codes. The numeric values of these two types match up, although aclstat3 omits some codes that are not relevant to the NFS_ACL protocol.

```
enum aclstat3 {
    ACL3_OK             = 0,
    ACL3ERR_PERM        = 1,
    ACL3ERR_NOENT       = 2,
    ACL3ERR_IO          = 5,
    ACL3ERR_ACCES       = 13,
    ACL3ERR_INVAL       = 22,
    ACL3ERR_NOSPC       = 28,
    ACL3ERR_ROFS        = 30,
    ACL3ERR_DQUOT       = 69,
    ACL3ERR_STALE       = 70,
    ACL3ERR_BADHANDLE   = 10001,
    ACL3ERR_NOTSUPP     = 10004,
    ACL3ERR_SERVERFAULT = 10006,
    ACL3ERR_JUKEBOX     = 10008
};
```

These status codes carry the following meanings:

ACL3_OK Indicates the call completed successfully.

ACL3ERR_PERM Not owner. The operation was not allowed because the caller is either not a privileged user (root) or not the owner of the target of the operation.

ACL3ERR_NOENT No such file or directory. The file or directory name specified does not exist.

ACL3ERR_IO I/O error. A hard error (for example, a disk error) occurred while processing the requested operation.

ACL3ERR_ACCES Permission denied. The caller does not have the correct permission to perform the requested operation. Contrast this with NFS3ERR_PERM, which restricts itself to owner or privileged user permission failures.

ACL3ERR_INVAL An invalid or unsupported argument was specified for procedure.

ACL3ERR_NOSPC No space left on device. The operation would have caused the server's file system to exceed its limit.

ACL3ERR_ROFS Read-only file system. A modifying operation was attempted on a read-only file system.

ACL3ERR_DQUOT Resource (quota) hard limit exceeded. The user's resource limit on the server has been exceeded.

ACL3ERR_STALE Invalid file handle. The file handle given in the arguments was invalid. The file referred to by that file handle no longer exists or access to it has been revoked.

ACL3ERR_BADHANDLE Illegal NFS file handle. The file handle failed internal consistency checks.

ACL3ERR_NOTSUPP Operation is not supported.

ACL3ERR_SERVERFAULT An error occurred on the server which does not map to any of the legal NFS version 3 protocol error values. The client should translate this into an appropriate error. UNIX clients may choose to translate this to EIO.

ACL3ERR_JUKEBOX The server initiated the request, but was not able to complete it in a timely fashion. The client should wait and then try the request with a new RPC transaction ID. For example, this error should be returned from a server that supports hierarchical storage and receives a request to process a file that has been migrated. In this case, the server should start the immigration process and respond to client with this error.

6.3. Server Procedures

6.3.1. Procedure 0: NULL - No Operation

6.3.1.1. ARGUMENTS

```
void;
```

6.3.1.2. RESULTS

```
void;
```

6.3.1.3. DESCRIPTION

This is the usual NULL procedure with a void argument and void result.

6.3.1.4. IMPLEMENTATION

It is important that this procedure do no work at all so that clients can use it to measure the overhead of processing a service request. By convention, the NULL procedure should never require any authentication. A server implementation may choose to ignore this convention, if responding to the NULL procedure call acknowledges the existence of a resource to an unauthenticated client.

6.3.1.5. ERRORS

Since the NULL procedure takes no argument and returns no result, it can not return an NFS or NFS_ACL error status code. However, some server implementations may return RPC errors based on security or authentication policy settings.

6.3.2. Procedure 1: GETACL - Retrieve an Access Control List

6.3.2.1. ARGUMENTS

```
struct GETACL3args {
    nfs_fh3 fh;
    unsigned int mask;
};
```

6.3.2.2. RESULTS

```
struct GETACL3resok {
    post_op_attr attr;
    secattr acl;
};

struct GETACL3resfail {
    post_op_attr attr;
};

union GETACL3res switch (aclstat3 status) {
case ACL3_OK:
    GETACL3resok resok;
default:
    GETACL3resfail resfail;
};
```

6.3.2.3. DESCRIPTION

The GETACL procedure retrieves Access Control List information associated with the file system object specified by the GETACL3args.fh field. The client obtains this file handle using one of the NFS version 2 LOOKUP, CREATE, MKDIR, SYMLINK, MKNOD, or READDIRPLUS procedures, or the MOUNT service, as described in [RFC1813].

The GETACL3args.mask field specifies which information is to be returned in the response:

* If the NA_ACL bit is set, the server fills in the object's access ACL.

* If the NA_ACLCNT bit is set, the server fills in the number of ACEs that are in the object's access ACL.

* If the NA_DFACL bit is set, the server fills in the object's default ACL.

* if the NA_DFACLCNT bit is set, the server fills in the number of ACEs that are in the object's default ACL.

If the GETACL procedure is successful, the server sets the GETACL3res.status field to ACL3_OK. It fills in the GETACL3resok.attr field with the file object's post operation file attributes, as detailed in [RFC1813]. Lastly, it fills in the GETACL3resok.acl field with two counted arrays of Access Control Entries (ACEs).

Otherwise, GETACL3res.status contains an error status on failure and no other results are returned.

6.3.2.4. IMPLEMENTATION

When GETACL3args.fh represents a file object that does not currently have an ACL associated with it or does not implement support for ACLs, the server responds by returning a manufactured minimal NFS ACL that reflects the current owner, group, and mode bits of the object (see Section 3.5.4.2).

6.3.2.5. ERRORS

* ACL3ERR_IO

* ACL3ERR_STALE

* ACL3ERR_BADHANDLE

* ACL3ERR_SERVERFAULT

* ACL3ERR_JUKEBOX

6.3.3. Procedure 2: SETACL - Set or replace an Access Control List

6.3.3.1. ARGUMENTS

```
struct SETACL3args {
    nfs_fh3 fh;
    secattr acl;
};
```

6.3.3.2. RESULTS

```
struct SETACL3resok {
    post_op_attr attr;
};

struct SETACL3resfail {
    post_op_attr attr;
};

union SETACL3res switch (aclstat3 status) {
case ACL3_OK:
    SETACL3resok resok;
default:
    SETACL3resfail resfail;
};
```

6.3.3.3. DESCRIPTION

The SETACL procedure replaces the Access Control Lists associated with the file system object specified by the SETACL3args.fh field with the ACLs specified by the SETACL3args.acl field. The client obtains the file handle using one of the NFS version 3 LOOKUP, CREATE, MKDIR, MKNOD, SYMLINK, or READDIRPLUS procedures, or the MOUNT service, as described in [RFC1813].

To remove extended access control from a file object, a client uses SETACL to replace the object’s ACL with a minimal NFS ACL (see Section 3.5.4.2).

If the SETACL procedure is successful, the server sets the SETACL3res.status field to ACL3_OK and fills in the SETACL3resok.attr field with the file object's post operation file attributes, as detailed in [RFC1813].

Otherwise, SETACL3res.status contains an error status on failure and no other results are returned.

6.3.3.4. IMPLEMENTATION

On success, the server does not send the reply until the ACL change is durable locally.

Changing a file object's ACL changes the object's mtime. The mtime change is reflected in the attributes returned in the SETACL response.

A high-quality server implementation ensures that a GETACL procedure running concurrently with a SETACL procedure does not return partially updated (torn) ACL contents. However, a failed SETACL may partially change a file's ACLs.

When SETACL3args.fh represents a file object that does not implement support for ACLs, the server responds by setting SETACL3res.status to ACL3ERR_NOTSUPP.

When SETACL3args.acl does not contain at least the minimal set of ACEs (as described in Section 3.5.4.2), the server responds by setting SETACL3res.status to ACL3ERR_INVAL.

6.3.3.5. ERRORS

* ACL3ERR_PERM
* ACL3ERR_IO
* ACL3ERR_ACCES
* ACL3ERR_INVAL
* ACL3ERR_NOSPC
* ACL3ERR_ROFS
* ACL3ERR_DQUOT
* ACL3ERR_STALE

* ACL3ERR_BADHANDLE

* ACL3ERR_SERVERFAULT

* ACL3ERR_JUKEBOX

6.3.4. Procedure 3: GETXATTRDIR - Get named attribute directory

6.3.4.1. ARGUMENTS

```
struct GETXATTRDIR3args {
    nfs_fh3 fh;
    bool create;
};
```

6.3.4.2. RESULTS

```
struct GETXATTRDIR3resok {
    nfs_fh3 fh;
    post_op_attr attr;
};

union GETXATTRDIR3res switch (aclstat3 status) {
case ACL3_OK:
    GETXATTRDIR3resok resok;
default:
    void;
};
```

6.3.4.3. DESCRIPTION

Section 5.3 of [RFC8881] defines a set of generic file attributes known as "named attributes". The GETXATTRDIR procedure extends this facility into the NFSv3 protocol.

The GETXATTRDIR procedure obtains the file handle of the named attribute directory associated with the file handle in the GETXATTRDIR3args.fh field. This directory contains only objects of type NF3REG.

If the GETXATTRDIR procedure is successful, the server sets the GETXATTRDIR3res.status field to ACL3_OK. It fills in the GETXATTRDIR3resok.fh field with a file handle that the client may use to look up the target file’s named attributes. It fills in the GETXATTRDIR3resok.attr field with the name attribute directory’s current file attributes, as detailed in [RFC1813].

Using the file handle returned in GETXATTRDIR3resok.fh, a client can utilize the READDIR and LOOKUP procedures to obtain file handles for the named attributes associated with the target file system object.

If the target file object does not currently have a named attribute directory associated with it and the GETXATTRDIR3args.create boolean field is set to false, the server returns ACL3ERR_NOENT. If the target file object does not currently have a named attribute directory associated with it and the GETXATTRDIR3args.create boolean field is set to true, the server attempts to create the named attribute directory before returning a result. If the target file currently has a named attribute directory associated with it and the GETXATTRDIR3args.create boolean is set to true, the server returns the file handle of that named attribute directory.

If the RPC user does not have read access to the target file, or if the GETXATTRDIR operation is to create a named attribute directory and the RPC user does not have permission to do so, the server returns ACL3_ACCES in the GETXATTRDIR3.status field.

If the target file handle designates an object not of type NF3REG or NF3DIR, the server returns the value ACL3ERR_INVAL in the GETXATTRDIR3.status field. Neither named attributes nor named attribute directories have their own named attributes.

Note: This operation is equivalent to the NFSv4 OPENATTR operation as specified in Section 16.17 of [RFC7530] and Section 18.17 of [RFC8881].

6.3.4.4. IMPLEMENTATION

Server implementers are free to choose not to implement this procedure. In this case, the server returns the RPC-level error PROC_UNAVAIL.

If the server implementation does implement the GETXATTRDIR procedure but the shared file system containing the file object specified by the file handle in the GETXATTRDIR3args.fh field does not support named attributes, the server returns ACL3ERR_NOTSUPP in the GETXATTRDIR3.status field.

6.3.4.5. ERRORS

* ACL3ERR_PERM
* ACL3ERR_NOENT
* ACL3ERR_IO

* ACL3ERR_ACCES
* ACL3ERR_INVAL
* ACL3ERR_NOSPC
* ACL3ERR_ROFS
* ACL3ERR_STALE
* ACL3ERR_NOTSUPP
* ACL3ERR_SERVERFAULT
* ACL3ERR_JUKEBOX

7. Implementation Issues

7.1. Permission issues

The NFS protocol, strictly speaking, does not define the permission checking used by NFS servers. However, it is expected that an NFS server will do normal operating system permission checking using AUTH_UNIX style authentication as the basis of its protection mechanism, or another stronger form of authentication such as RPCSEC_GSS. With AUTH_UNIX authentication, the server gets the client's effective uid, effective gid, and groups on each call and uses them to check permission. These are the so-called UNIX credentials.

Using uid and gid implies that the client and server share the same uid list. Every server and client pair must have the same mapping from user to uid and from group to gid. Since every client can also be a server, this tends to imply that the whole network shares the same uid/gid space. If this is not the case, then it usually falls upon the server to perform some custom mapping of credentials from one authentication domain into another. A discussion of techniques for managing a shared user space or for providing mechanisms for user ID mapping is beyond the scope of this specification.

In POSIX-based operating systems, a particular user (on UNIX, the uid 0) has access to all files, no matter what permission and ownership they have. This superuser permission may not be allowed on the server, since anyone who can become superuser on their client could gain access to all remote files. A POSIX-based NFS server by default maps uid 0 to a distinguished value (for instance, UID_NOBODY), as well as mapping the groups list, before doing its access checking. A server implementation may provide a mechanism to change this mapping.

7.2. Duplicate Request Cache

The typical NFS protocol failure recovery model uses client time-out and retry to handle server crashes, network partitions, and lost server replies. A retried request is referred to as a duplicate of the original.

When used in a file server context, the term idempotent can be used to distinguish between operation types. An idempotent request is one that a server can perform more than once with equivalent results (though it may in fact change, as a side effect, the access time on a file, say for READ). Some NFS operations are obviously non-idempotent. They cannot be reprocessed without special attention simply because they may fail if tried a second time. A CREATE request, for example, can be used to create a file for which the owner does not have write permission. A duplicate of this request cannot succeed if the original succeeded. Likewise, a file can be removed only once.

The side effects caused by performing a duplicate non-idempotent request can be destructive. A duplicate file truncation can result in lost writes. It is the inherent stateless design of the NFS protocol on top of an unreliable RPC transport that yields the possibility of destructive replays of non-idempotent requests. Even in an implementation of the NFS protocol over a reliable connection-oriented transport, a connection break with automatic reestablishment requires duplicate request processing: the client retransmits requests that were pending before the connection loss, and the server needs to recognize and deal with potential duplicate non-idempotent requests.

Most NFS server implementations maintain a cache of recent requests, called the duplicate request cache, for recognizing duplicate non-idempotent requests. If the server receives a request and recognizes it as a duplicate of a recently completed request, the server returns the original completion status instead of processing the duplicate request again.

A description of an early implementation of a duplicate request cache can be found in [Juszczak].

For all versions of the NFS_ACL protocol, the SETACL procedure is considered to be non-idempotent.

7.3. Caching Policies

The NFS protocol does not define a policy for caching on the client or server. In particular, there is no support for strict cache consistency between a client and server, nor between different clients.

The NFS_ACL protocol does not mandate a specific caching policy for ACLs or information retrieved via the ACCESS procedure. However, a high-quality client implementation that seeks good performance might choose to revalidate cached access control information with the same regularity that it invalidates normal file attributes.

8. XDR Protocol Definition

This section contains a description of the core features of the NFS_ACL protocol, version 2 and 3, expressed in the XDR language [RFC4506].

This description is provided in a way that makes it simple to extract into ready-to-compile form. The reader can apply the following shell script to this document to produce a machine-readable XDR description of the NFS_ACL protocol.

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

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 the following to extract an XDR description file:

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

8.1. Code Component License

Code components extracted from this document must include the following license text. When the extracted XDR code is combined with other complementary XDR code which itself has an identical license, only a single copy of the license text need be preserved.

```
/// /*
///  * Copyright (c) 2024 IETF Trust and the persons
///  * identified as authors of the code.  All rights reserved.
///  *
///  * The authors of the code are:
///  * Oracle
///  *
///  * Redistribution and use in source and binary forms, with
```

```
   ///  * or without modification, are permitted provided that the
   ///  * following conditions are met:
   ///  *
   ///  * - Redistributions of source code must retain the above
   ///  *   copyright notice, this list of conditions and the
   ///  *   following disclaimer.
   ///  *
   ///  * - 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.
   ///  *
   ///  * - 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.
   ///  */
   ///
   /// const NFS_ACL_MAX_ENTRIES = 1024;
   ///
   /// typedef unsigned int uid;
   /// typedef unsigned short o_mode;
   ///
   /// /*
   ///  * This is the format of an ACL which is passed over the network.
   ///  */
   /// struct aclent {
   ///     int type;
   ///     uid id;
   ///     o_mode perm;
   /// };
   ///
```

```
   /// /*
   ///  * The values for the type element of the aclent structure.
   ///  */
   /// const NA_USER_OBJ = 0x1;            /* object owner */
   /// const NA_USER = 0x2;                /* additional users */
   /// const NA_GROUP_OBJ = 0x4;           /* owning group of the object */
   /// const NA_GROUP = 0x8;               /* additional groups */
   /// const NA_CLASS_OBJ = 0x10;          /* file group class and mask entry */
   /// const NA_OTHER_OBJ = 0x20;          /* other entry for the object */
   /// const NA_ACL_DEFAULT = 0x1000;      /* default flag */
   ///
   /// /*
   ///  * The bit field values for the perm element of the aclent
   ///  * structure.  The three values can be combined to form any
   ///  * of the 8 combinations.
   ///  */
   /// const NA_READ = 0x4;                /* read permission */
   /// const NA_WRITE = 0x2;               /* write permission */
   /// const NA_EXEC = 0x1;                /* exec permission */
   ///
   /// /*
   ///  * This is the structure which contains the ACL entries for a
   ///  * particular entity.  It contains the ACL entries which apply
   ///  * to this object plus any default ACL entries which are
   ///  * inherited by its children.
   ///  *
   ///  * The values for the mask field are defined below.
   ///  */
   /// struct secattr {
   ///     unsigned int mask;
   ///     int aclcnt;
   ///     aclent aclent<NFS_ACL_MAX_ENTRIES>;
   ///     int dfaclcnt;
   ///     aclent dfaclent<NFS_ACL_MAX_ENTRIES>;
   /// };
   ///
   /// /*
   ///  * The values for the mask element of the secattr struct as well
   ///  * as for the mask element in the arguments in the GETACL2 and
   ///  * GETACL3 procedures.
   ///  */
   /// const NA_ACL = 0x1;                 /* aclent contains a valid list */
   /// const NA_ACLCNT = 0x2;              /* number of entries in the aclent list */
   /// const NA_DFACL = 0x4;               /* dfaclent contains a valid list */
   /// const NA_DFACLCNT = 0x8;            /* number of entries in the dfaclent list */
   ///
   /// /*
   ///  * XDR data types inherited from the NFS version 2 protocol
```

```
   ///  */
   ///
   /// enum ftype {
   ///     NFNON = 0,
   ///     NFREG = 1,
   ///     NFDIR = 2,
   ///     NFBLK = 3,
   ///     NFCHR = 4,
   ///     NFLNK = 5,
   /// };
   ///
   /// const FHSIZE = 32;
   /// typedef opaque fhandle[FHSIZE];
   ///
   /// struct timeval {
   ///     unsigned int seconds;
   ///     unsigned int useconds;
   /// };
   ///
   /// struct fattr {
   ///     ftype        type;
   ///     unsigned int mode;
   ///     unsigned int nlink;
   ///     unsigned int uid;
   ///     unsigned int gid;
   ///     unsigned int size;
   ///     unsigned int blocksize;
   ///     unsigned int rdev;
   ///     unsigned int blocks;
   ///     unsigned int fsid;
   ///     unsigned int fileid;
   ///     timeval      atime;
   ///     timeval      mtime;
   ///     timeval      ctime;
   /// };
   ///
   /// /*
   ///  * ACL error codes; the numeric values match codes with the same
   ///  * name used in NFS version 2.
   ///  */
   /// enum aclstat2 {
   ///     ACL2_OK = 0,
   ///     ACL2ERR_PERM = 1,
   ///     ACL2ERR_NOENT = 2,
   ///     ACL2ERR_IO = 5,
   ///     ACL2ERR_ACCES = 13,
   ///     ACL2ERR_NOSPC = 28,
   ///     ACL2ERR_ROFS = 30,
```

```
   ///     ACL2ERR_DQUOT = 69,
   ///     ACL2ERR_STALE = 70,
   /// };
   ///
   /// /*
   ///  * NFS_ACL version 2 procedure arguments and results
   ///  */
   ///
   /// struct GETACL2args {
   ///     fhandle fh;
   ///     unsigned int mask;
   /// };
   ///
   /// struct GETACL2resok {
   ///     struct nfsfattr attr;
   ///     secattr acl;
   /// };
   ///
   /// union GETACL2res switch (enum nfsstat status) {
   /// case ACL2_OK:
   ///     GETACL2resok resok;
   /// default:
   ///     void;
   /// };
   ///
   /// struct SETACL2args {
   ///     fhandle fh;
   ///     secattr acl;
   /// };
   ///
   /// struct SETACL2resok {
   ///     struct nfsfattr attr;
   /// };
   ///
   /// union SETACL2res switch (enum nfsstat status) {
   /// case ACL2_OK:
   ///     SETACL2resok resok;
   /// default:
   ///     void;
   /// };
   ///
   /// struct GETATTR2args {
   ///     fhandle fh;
   /// };
   ///
   /// struct GETATTR2resok {
   ///     struct nfsfattr attr;
   /// };
```

```
   ///
   /// union GETATTR2res switch (enum nfsstat status) {
   /// case ACL2_OK:
   ///     GETATTR2resok resok;
   /// default:
   ///     void;
   /// };
   ///
   /// struct ACCESS2args {
   ///     fhandle fh;
   ///     uint32 access;
   /// };
   ///
   /// const ACCESS2_READ = 0x1;           /* read data or readdir a directory */
   /// const ACCESS2_LOOKUP = 0x2;         /* lookup a name in a directory */
   /// const ACCESS2_MODIFY = 0x4;         /* rewrite existing file data or */
   ///                                     /* modify existing directory entries */
   /// const ACCESS2_EXTEND = 0x8;         /* write new data or add directory entries */
   /// const ACCESS2_DELETE = 0x10;        /* delete existing directory entry */
   /// const ACCESS2_EXECUTE = 0x20;       /* execute file (no meaning for a directory) *
/
   ///
   /// struct ACCESS2resok {
   ///     struct nfsfattr attr;
   ///     uint32 access;
   /// };
   ///
   /// union ACCESS2res switch (enum nfsstat status) {
   /// case ACL2_OK:
   ///     ACCESS2resok resok;
   /// default:
   ///     void;
   /// };
   ///
   /// /*
   ///  * This is the definition for the GETXATTRDIR procedure which applies
   ///  * to NFS Version 2 files.
   ///  */
   /// struct GETXATTRDIR2args {
   ///     fhandle fh;
   ///     bool create;
   /// };
   ///
   /// struct GETXATTRDIR2resok {
   ///     fhandle fh;
   ///     struct nfsfattr attr;
   /// };
   ///
   /// union GETXATTRDIR2res switch (enum nfsstat status) {
```

```
   /// case ACL2_OK:
   ///     GETXATTRDIR2resok resok;
   /// default:
   ///     void;
   /// };
   ///
   /// /*
   ///  * XDR data types inherited from the NFS version 3 protocol
   ///  */
   ///
   /// typedef unsigned hyper uint64;
   /// typedef unsigned long uint32;
   /// typedef uint64 fileid3;
   /// typedef uint32 uid3;
   /// typedef uint32 gid3;
   /// typedef uint64 size3;
   /// typedef uint32 mode3;
   ///
   /// enum ftype3 {
   ///     NF3REG    = 1,
   ///     NF3DIR    = 2,
   ///     NF3BLK    = 3,
   ///     NF3CHR    = 4,
   ///     NF3LNK    = 5,
   ///     NF3SOCK   = 6,
   ///     NF3FIFO   = 7
   /// };
   ///
   /// struct specdata3 {
   ///     uint32     specdata1;
   ///     uint32     specdata2;
   /// };
   ///
   /// const NFS3_FHSIZE = 64;
   ///
   /// struct nfs_fh3 {
   ///     opaque       data<NFS3_FHSIZE>;
   /// };
   ///
   /// struct nfstime3 {
   ///     uint32   seconds;
   ///     uint32   nseconds;
   /// };
   ///
   /// struct fattr3 {
   ///     ftype3     type;
   ///     mode3      mode;
   ///     uint32     nlink;
```

```
   ///     uid3       uid;
   ///     gid3       gid;
   ///     size3      size;
   ///     size3      used;
   ///     specdata3  rdev;
   ///     uint64     fsid;
   ///     fileid3    fileid;
   ///     nfstime3   atime;
   ///     nfstime3   mtime;
   ///     nfstime3   ctime;
   /// };
   ///
   /// union post_op_attr switch (bool attributes_follow) {
   /// case TRUE:
   ///     fattr3   attributes;
   /// case FALSE:
   ///     void;
   /// };
   ///
   /// /*
   ///  * ACL error codes; the numeric values match codes with the same
   ///  * name used in NFS version 3.
   ///  */
   /// enum aclstat3 {
   ///     ACL3_OK = 0,
   ///     ACL3ERR_PERM = 1,
   ///     ACL3ERR_NOENT = 2,
   ///     ACL3ERR_IO = 5,
   ///     ACL3ERR_ACCES = 13,
   ///     ACL3ERR_INVAL = 22,
   ///     ACL3ERR_NOSPC = 28,
   ///     ACL3ERR_ROFS = 30,
   ///     ACL3ERR_DQUOT = 69,
   ///     ACL3ERR_STALE = 70,
   ///     ACL3ERR_BADHANDLE = 10001,
   ///     ACL3ERR_NOTSUPP = 10004,
   ///     ACL3ERR_SERVERFAULT = 10006,
   ///     ACL3ERR_JUKEBOX = 10008,
   /// };
   ///
   /// /*
   ///  * NFS_ACL version 3 procedure arguments and results
   ///  */
   ///
   /// struct GETACL3args {
   ///     nfs_fh3 fh;
   ///     unsigned int mask;
   /// };
```

```
   ///
   /// struct GETACL3resok {
   ///     post_op_attr attr;
   ///     secattr acl;
   /// };
   ///
   /// struct GETACL3resfail {
   ///     post_op_attr attr;
   /// };
   ///
   /// union GETACL3res switch (aclstat3 status) {
   /// case ACL3_OK:
   ///     GETACL3resok resok;
   /// default:
   ///     GETACL3resfail resfail;
   /// };
   ///
   /// struct SETACL3args {
   ///     nfs_fh3 fh;
   ///     secattr acl;
   /// };
   ///
   /// struct SETACL3resok {
   ///     post_op_attr attr;
   /// };
   ///
   /// struct SETACL3resfail {
   ///     post_op_attr attr;
   /// };
   ///
   /// union SETACL3res switch (aclstat3 status) {
   /// case ACL3_OK:
   ///     SETACL3resok resok;
   /// default:
   ///     SETACL3resfail resfail;
   /// };
   ///
   /// /*
   ///  * This is the definition for the GETXATTRDIR procedure which
   ///  * applies to NFS Version 3 files.
   ///  */
   /// struct GETXATTRDIR3args {
   ///     nfs_fh3 fh;
   ///     bool create;
   /// };
   ///
   /// struct GETXATTRDIR3resok {
   ///     nfs_fh3 fh;
```

```
   ///     post_op_attr attr;
   /// };
   ///
   /// union GETXATTRDIR3res switch (aclstat3 status) {
   /// case ACL3_OK:
   ///     GETXATTRDIR3resok resok;
   /// default:
   ///     void;
   /// };
   ///
   /// /*
   ///  * Share the port with the NFS service.
   ///  */
   /// const NFS_ACL_PORT = 2049;
   ///
   /// program NFS_ACL_PROGRAM {
   ///     version NFS_ACL_V2 {
   ///         void
   ///             ACLPROC2_NULL(void) = 0;
   ///         GETACL2res
   ///             ACLPROC2_GETACL(GETACL2args) = 1;
   ///         SETACL2res
   ///             ACLPROC2_SETACL(SETACL2args) = 2;
   ///         GETATTR2res
   ///             ACLPROC2_GETATTR(GETATTR2args) = 3;
   ///         ACCESS2res
   ///             ACLPROC2_ACCESS(ACCESS2args) = 4;
   ///         GETXATTRDIR2res
   ///             ACLPROC2_GETXATTRDIR(GETXATTRDIR2args) = 5;
   ///     } = 2;
   ///     version NFS_ACL_V3 {
   ///         void
   ///             ACLPROC3_NULL(void) = 0;
   ///         GETACL3res
   ///             ACLPROC3_GETACL(GETACL3args) = 1;
   ///         SETACL3res
   ///             ACLPROC3_SETACL(SETACL3args) = 2;
   ///         GETXATTRDIR3res
   ///             ACLPROC3_GETXATTRDIR(GETXATTRDIR3args) = 3;
   ///     } = 3;
   /// } = 100227;
```

9. Implementation Status

> | This section is to be removed before publishing this document
> | as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

9.1. Solaris NFS server and client

Organization: Oracle

URL: https://www.oracle.com (https://www.oracle.com)

Maturity: Complete.

Coverage: All procedures are implemented.

Licensing: CDDL

Implementation experience:

9.2. Linux NFS server and client

Organization: The Linux Foundation

URL: https://www.kernel.org (https://www.kernel.org)

Maturity: Complete.

Coverage: All procedures except GETXATTRDIR are implemented in both versions of the protocol.

Licensing: GPLv2

Implementation experience: The initial Linux implementation of the NFS_ACL protocol is described in [Gruenbacher], and subsequent modifications can be found in the Linux kernel source code repository [Linux].

[Gruenbacher] notes several minor differences between the Linux and Solaris implementations of ACLs, and remarks that: > Solaris ACLs are based on an earlier draft of POSIX 1003.1e, > so its handling of the mask ACL entry is slightly different > than in draft 17 for ACLs with only four ACL entries. This > is a corner case that occurs only rarely, so the semantic > differences may not be noticeable.

10. Security Considerations

An attacker can alter the content of an ACL as it transits an open network, giving the attacker access to file content that the ACL is supposed to protect.

Therefore, implementations of NFS_ACL should protect the integrity of ACL content when it transits a network. The use of an integrity-preserving transport layer security service, such as the GSS Kerberos integrity service, is strongly recommended.

11. IANA Considerations

In accordance with Section 13 of [RFC5531], the editor requests that IANA update the entry for the NFS ACL service in the RPC Program Numbers registry to add the current document as a Reference.

12. References

12.1. Normative References

[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/rfc/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/rfc/rfc4506>.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, DOI 10.17487/RFC5531, May 2009, <https://www.rfc-editor.org/rfc/rfc5531>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[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/rfc/rfc8174>.

12.2. Informative References

[Gruenbacher]
Grnbacher, A., "POSIX Access Control Lists on Linux", Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference, pp. 259-272, ISBN 1-931971-11-0, January 2003.

[I-D.ietf-nfsv4-posix-acls]
Macklem, R., "POSIX Draft ACL support for Network File System Version 4, Minor Version 2", Work in Progress, Internet-Draft, draft-ietf-nfsv4-posix-acls-01, 8 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nfsv4-posix-acls-01>.

[IEEE] Institute of Electrical and Electronics Engineers, "IEEE 1003.1e and 1003.2c: Draft Standard for Information Technology-- Portable Operating System Interface (POSIX)-- Part 1: System Application Program Interface (API) and Part 2: Shell and Utilities, draft 17", January 1997.

[Juszczak] Juszcak, C., "Improving the Performance and Correctness of an NFS Server", USENIX Conference Proceedings, USENIX Association, Berkeley, CA, pp. 53-63, January 1989.

[Linux] "Linux kernel source code", n.d., <https://www.kernel.org>.

[OpenSolaris]
"Archived OpenSolaris source code", n.d., <https://github.com/kofemann/opensolaris>.

[POSIX] Institute of Electrical and Electronics Engineers, "IEEE Std 1003.1-2001 (Open Group Technical Standard, Issue 6), Standard for Information Technology-- Portable Operating System Interface (POSIX)", ISBN 0-7381-3010-9, 2001.

[RFC1094] Nowicki, B., "NFS: Network File System Protocol specification", RFC 1094, DOI 10.17487/RFC1094, March 1989, <https://www.rfc-editor.org/rfc/rfc1094>.

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <https://www.rfc-editor.org/rfc/rfc1813>.

[RFC1833] Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, DOI 10.17487/RFC1833, August 1995, <https://www.rfc-editor.org/rfc/rfc1833>.

[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/rfc/rfc7530>.

[RFC8881] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <https://www.rfc-editor.org/rfc/rfc8881>.

Appendix A. Source Material

The on-the-wire protocol described here is intended to match existing de facto implementations of NFS_ACL.

The source for the XDR specification provided in this document is the nfs_acl.x file as found in published versions of the OpenSolaris source code base [OpenSolaris], an open source descendant of Solaris.

However, there are a few changes to the protocol as it was originally described in the OpenSolaris source code base.

A.1. Redaction of NFS_ACL Version 4

Version 4 of NFS_ACL is described in the original nfs_acl.x source file this way:

> This is a transitional interface to enable Solaris NFSv4 clients to manipulate ACLs on Solaris servers until the spec is complete enough to implement this inside the NFSv4 protocol itself. NFSv4 does handle extended attributes in-band.

Because the two non-NULL procedures in this version of the NFS_ACL protocol were used only as part of a Solaris a prototype and there are no other implementations of NFS_ACL version 4, it is not included in the protocol description appearing in this document.

A.2. Extension of NFS_ACL

Extension of this legacy protocol is out of scope for an Informational document whose purpose is to describe existing implementations.

A.3. Code Compilation Requirements

The original nfs_acl.x file that appears in the OpenSolaris code base did not compile using the widely-available rpcgen tool).

* The file does not include a definition of the ACL2_OK or ACL3_OK constants used in definitions of result unions.

* The file does not include definitions of NFS protocol elements that are shared with the NFS_ACL protocol, such as fhandle and post_op_attr.

The XDR specification provided in this document rectifies those omissions to provide a complete and compilable XDR language description of the NFS_ACL protocol.

Acknowledgments

The editor is grateful to Bill Baker, Wim Coekaerts, Andreas Gruenbacher, Rick Macklem, Greg Marsden, Martin Thomson, Rob Thurlow, and Jim Wright for their input and support.

Special thanks to Area Director Gorry Fairhurst, NFSV4 Working Group Chairs Brian Pawlowski and Christopher Inacio, and NFSV4 Working Group Secretary Thomas Haynes for their patience, guidance, and oversight.

Author’s Address

Chuck Lever (editor)
Independent
United States of America
Email: cel-ietf@chucklever.net