

More ACLs For Linux

File Sharing Across Operating System Boundaries

Andreas Grünbacher
SUSE Labs, Novell

August 2, 2010



Novell.[®]

Outline

- Project History
- Motivation and Existing Approaches
- File Permission Models
 - The POSIX File Permission Model
 - POSIX ACLs
 - NFSv4 ACLs
- Status

Project History

Project History

- ACL workshop with F. Bruce Fields and others in Broomfield/CO in 2006
- Native NFSv4 ACL project introduced in 2006 at the SuSE Labs Conference in Harrachov
- Presented at linux.conf.au in 2007
- “ACL hack month” at SGI Melbourne
- Shipped in SGI's Infinite Storage NAS product in 2008

- Picked up again at the end of 2009 by Aneesh Kumar/LTC
- Renamed to Richacl in 2010
- “ACL hack week” in Bangalore in June 2010
- Being presented at Ottawa Linux Symposium this week (Greg Banks/EvoStor)

Motivation and Existing Approaches

Operating Systems In Common Use

- Common operating systems
 - Linux
 - Windows
 - Some remaining UNIXes
- Operating Systems are typically mixed
 - Several different file permission models
 - Interoperability is important, also for file sharing!

File Permission Models

- The POSIX 1003.1 File Permission Model
 - Process classes: owner, group, other
 - Permissions: read, write, and search/execute
- POSIX 1003.1e draft 17 ACLs
 - ACL entries for additional users and groups
 - Still only read, write, and search/execute
- Windows ACLs
 - An ACL entry can *allow* or *deny* something
 - Additional permissions: append, take ownership, ...
 - Windows NT: Create-time inheritance
 - Since Windows 2000: Automatic Inheritance
 - Since Windows Vista: *OwnerRights* ACL entries
- NFS Version 4 ACLs
 - Similar to Windows ACLs, but some weirdnesses removed, e.g., ACL entries for the *owner* and the *owning group*
 - NFSv4: Create-time inheritance; NFSv4.1: Automatic Inheritance

Which Problems Does This Create?

- Applications (and users) do not understand all the different permission models
- Mapping from one model to another is hard
 - Different features, permissions, semantics, ...
 - Intransparent behavior, surprising results, frustrated users
- Existing applications rely on “their” file permission model
 - For example, we cannot simply ignore the POSIX file permission bits

Existing Approaches (1)

Samba

- Can map between POSIX and NFS4 ACLs
 - Mapping to POSIX ACLs is lossy; “not good enough”
 - Lack of features like Automatic Inheritance
- Windows ACLs can be implemented inside Samba
 - Full features for everything going through Samba
 - Local processes and other protocols “locked out”

Linux NFSv4 Implementation

- Server maps between POSIX and NFS4 ACLs
- On the client side, the NFSv4 ACLs are directly exposed
 - Mapping to POSIX ACLs is lossy; “not good enough”
 - Protocol details like user/group ID mapping are exposed to user-space
 - Client does not know when it is actually dealing with POSIX ACLs

Existing Approaches (2)

IBM JFS2 (AIX), GPFS (Closed Source)

- Both support POSIX file permissions and Windows ACLs
- Either mode or ACL determines access (whichever is set last)
 - Works if files are not actually shared across operating systems
 - Is horrible otherwise

NetApp / EMC NAS

- Similar ACL support as IBM JFS2 and GPFS

Solaris ZFS

- Supports only the NFSv4 ACL model
- Is not fully POSIX compliant!

New Approach and Goal

Implement a better ACL model

- Enforce this model *natively*: locally and remotely
- Fully POSIX standard compliant
- Two-way mapping to Windows ACLs and NFSv4 ACLs
- One-way mapping from POSIX ACLs

Goal

- Make Linux the best choice for file serving and as a client

File Permission Models: The POSIX File Permission Model

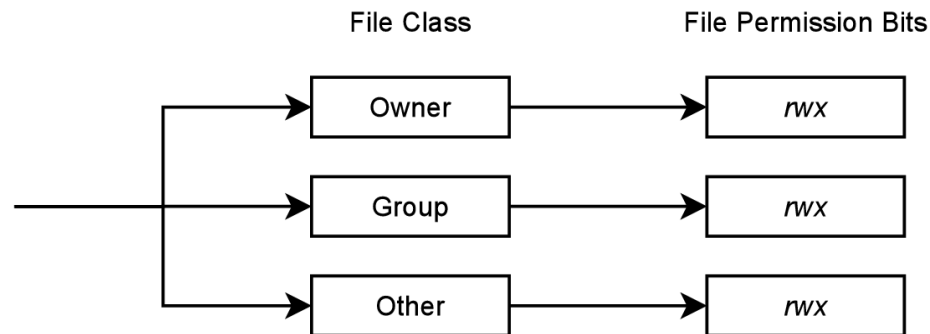
The POSIX File Permission Model (1)

“Participants” in file accesses

- Processes: effective user ID, effective group ID, supplementary group IDs
- Files: file permission bits, user ID, group ID

Three file classes

Three file permission bits for each class



The POSIX File Permission Model (2)

Extension mechanisms

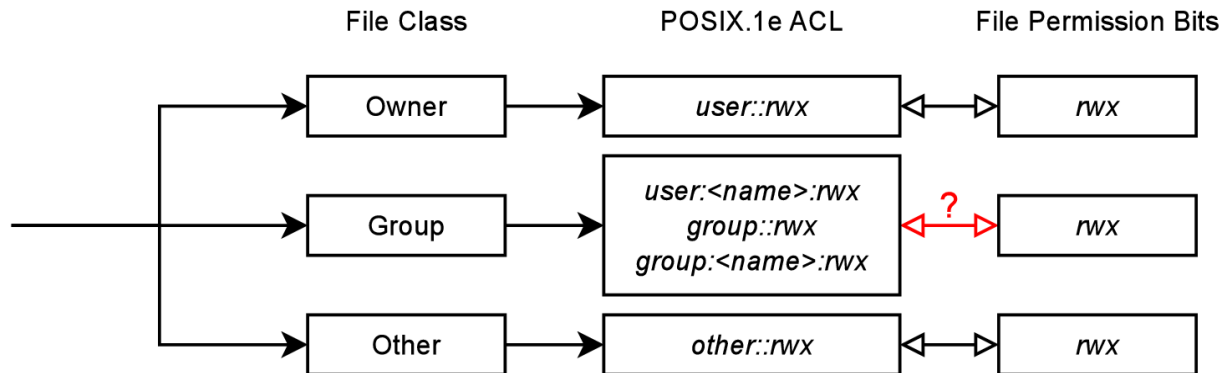
- *Additional* File Access Control Mechanisms
 - Can only further restrict the permissions of a process
- *Alternate* File Access Control Mechanisms
 - Can restrict or extend the permissions of a process
 - Must be enabled *explicitly* on a per-file basis (i.e., is off for new files)
 - Must be disabled when changing the file permission bits with `chmod()`
- The file group class may have *implementation-defined* members

File Permission Models:

POSIX ACLs

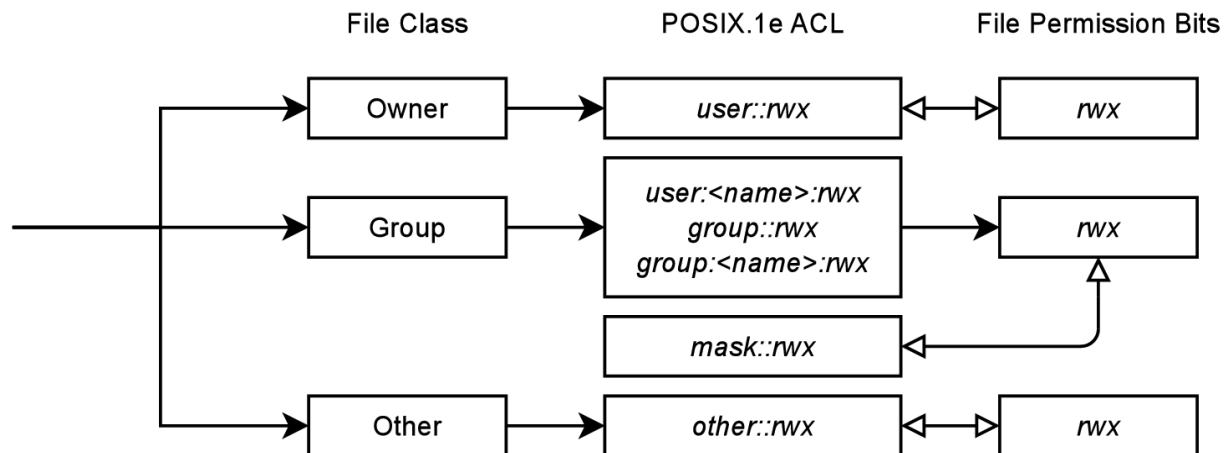
POSIX ACLs (1)

- Extension of the POSIX model, same concept of file classes
- Still only three file permission bits
- *Additional* file access control mechanism \Rightarrow only restrictive
- Additional user and group entries are in the file group class \Rightarrow limited by the group file permission bits – but how?



POSIX ACLs (2)

- The group class file permission bits *mask* the group class entries!



File Permission Models:

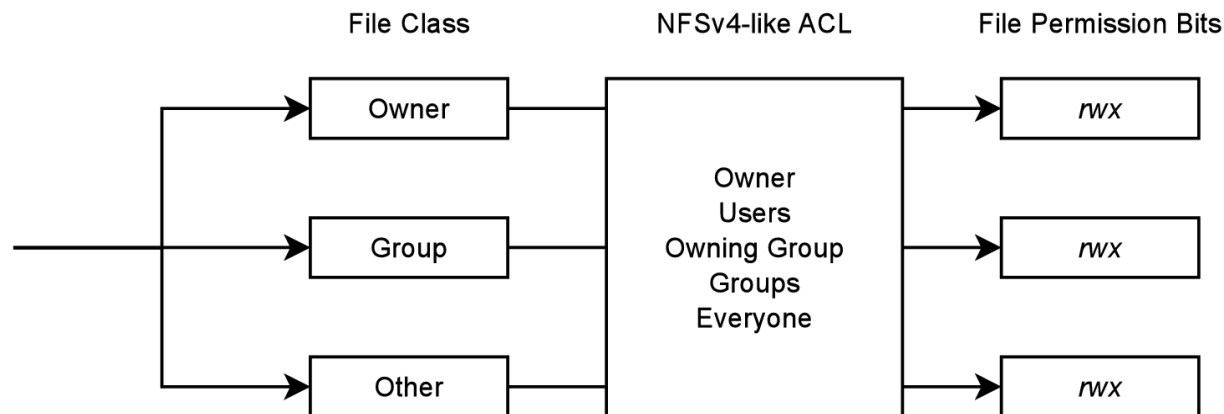
NFSv4 ACLs

NFSv4 ACLs

- File classes are not explicitly defined in the spec.
 - Needed for determining the effect of *chmod*
 - Fixable by defining a mapping
- More fine-grained file permissions (e.g., append vs. write)
 - Mapping between file permission bits and NFS4 ACL permissions
 - What is the effect of *chmod* on these permissions?
- Some permissions go beyond read, write, search/execute (e.g., change permissions, take ownership)
 - *Alternate* file access control mechanism
 - What is the effect of *chmod* on these permissions?

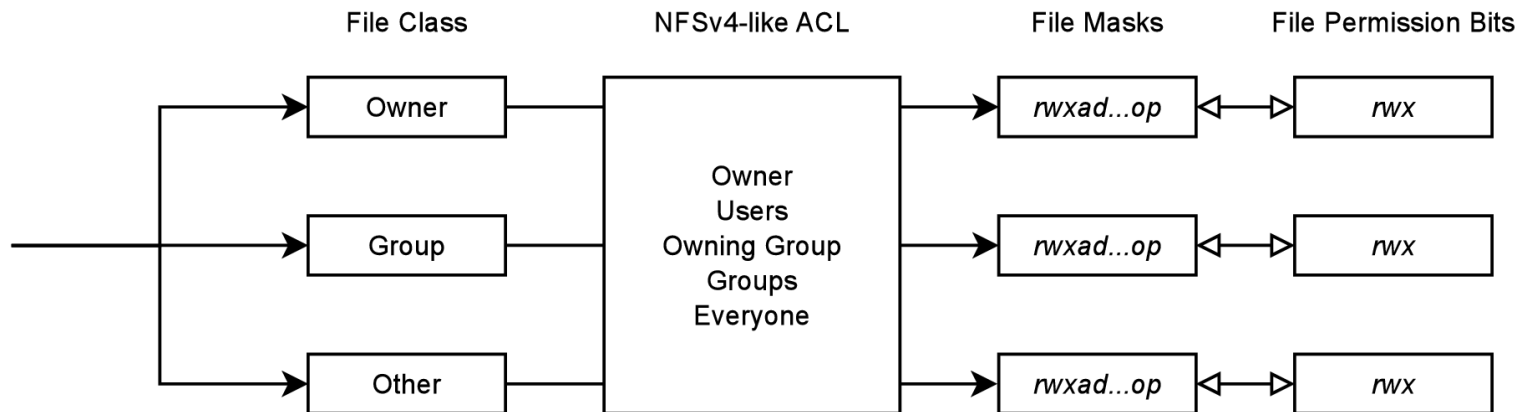
NFSv4 ACLs and POSIX (1)

- Instead of modifying the ACL upon *chmod*, we use masking as in POSIX ACLs:
 - A *chmod* only updates the file permission bits
 - Permission check algorithm checks the ACL and the file permission bits
- Examples: append, take ownership
 - The file permission bits won't allow to take ownership \Rightarrow ?



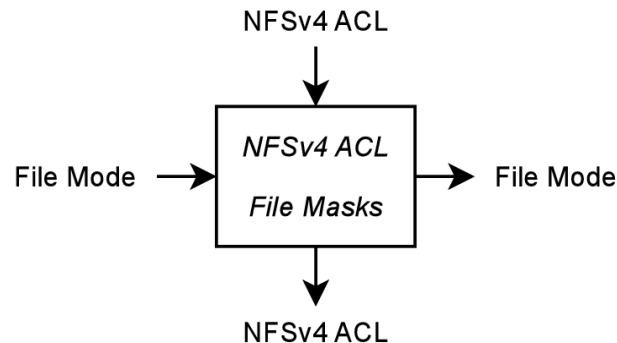
NFSv4 ACLs and POSIX (2)

- Solution: introduce file masks
 - Doing a *chmod* sets the file masks (and limits them to read, write, search/execute)
 - They can be set explicitly, too
 - Permission check algorithm checks the ACL and file masks



NFSv4 ACLs and POSIX (3)

- Protocols like CIFS or NFSv4 do not understand file masks
 - Algorithms for “applying” the file masks to the ACL exist
 - The “unmasked ACL + file masks” file permission check gives identical results as the “normal” NFSv4 ACL permission check



NFSv4 ACLs and POSIX (4)

- Much more thorough documentation available:
 - Ottawa Linux Symposium paper
 - Examples on the web page
 - Design document
 - Commit messages and source code comments

Status

Status

- Core code and ext4 support finished and stable
- A user-space utility and library exists
- NFSv4 client/server support exists (still needs more work)
- Samba code for nfs4acls exists (needs updating)
- Migration from POSIX ACLs to Richacls still under debate

- Rich ACLs:
<http://acl.bestbits.at/richacl/>

Thank you!

Novell.[®]

Unpublished Work of Novell, Inc. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary, and trade secret information of Novell, Inc. Access to this work is restricted to Novell employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of Novell, Inc. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. Novell, Inc., makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All Novell marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.



ZFS Quirks

- ZFS tries to represent “file masks” (as in richacl) as DENY ACEs
 - This breaks horribly when ACEs are reordered (the Windows ACL editor does that!)
- `chmod()` does not disable some permissions
 - For example, `delete_child` is not disabled
 - This violates the POSIX standard!
- Some things are absolutely bizarre
 - E.g., the `delete` vs. `delete_child` discussion and the `add_file` permission; see RFC 5661