Universal File System Extended Attributes Namespace

François Revol
Haiku Operating System
revol@free.fr

Abstract

The growing usage of file system extended attributes on many operating systems faces interoperability problems when trying to preserve them across multiple platforms. We propose a generic namespace design and idempotent mapping method to maintain an identical view of the global metadata namespace by each operating system. Additionally, we try to address the API and semantic incompatibilities with a higher level framework.

Keywords: metadata; interoperability; file-system; extended attributes; xattr; named streams; BFS; NTFS; SMB.

1. The Problem

File-system extended attributes, often abbreviated xattrs, or EAs, are a form of file metadata storage consisting of name-value pairs, and are used in many operating systems, under many forms and names (Resource Fork, Named Streams), for many purposes, either for security concerns (Access Control Lists, Proof Carrying), fallback for missing file-system properties (DOS attributes, POSIX "atime") or user-level applications (Leung et al., 2008), from early adopters like the BeOS, up to recent semantic desktops (Möller et al., 2007). At the file-system level they are usually handled as named raw data attached to the file's structure (inode), without any further semantic attached. Operating Systems then use them for their own purposes, like security, but most of them are left for application use as binary data without any interpretation by the Operating System, just as is done for file content itself.

However, various incarnations of EA concepts are usually incompatible with each other. Some have split namespaces for kernel-private data like Access Control Lists and user accessible metadata like Linux (attr(5) manpage), others have a single namespace using the reverse-DNS notation by convention like Mac OS X which maps the old HFS resource fork to EAs as `com.apple.ResourceFork`, some have typed values like BFS (Giampaolo, 1999) used in the BeOS and Haiku operating systems (Haiku), and the API to enumerate and access them aren't compatible. The POSIX drafts (1003.1e / 1003.2c Draft Standard 17) used for one of the several APIs in Linux has been withdrawn.

The growing usage of incompatible extended attributes conflicts with the need for interoperability, especially in OpenSource operating systems like GNU/Linux which now includes implementations for many foreign file-systems like FAT, NTFS, SMB, BFS and HFS. Each such implementation either uses a naive approach for mapping foreign extended attributes, leading to namespace pollution and name clashes (NTFS-3g), or a more complex but unilaterally-imposed (and thus not idempotent) mangling (Tridgell, 2005), when the file is moved across disks and systems, or sometimes just doesn't expose extended attributes at all due to lack of a clear mapping. A recent proposition for NFS extended attribute support on Linux already asserts incompatibility with the original IRIX implementation (Morris, 2009). Moreover, some file-systems, including `ext3` and even `ext4`, have severe limitations on the size available for EA storage (a single block per inode). Some old file-systems do not support extended attributes at all, for which several incompatible backing-store schemes have been devised, some being patented (French et al., 2008).

Even today, many GNU/Linux distributions do not enable "user_xattr" support for file-systems by default, and xattr-related features in many software like Samba are to be manually enabled. Haiku uses xattrs natively on its own file-system, but doesn't support them on most foreign file-systems, losing them when copying files over, and uses a custom extension to "ZIP" files to archive and transport them correctly. On the other hand, Mac OS X also natively uses them, but clutters foreign file-systems with `.DS_Store` and `._foobar` files even when xattrs could have been used instead. All this led software developers to avoid the use of xattrs even when they made sense up until now, also causing trouble in backup software that isn't aware of their existence. So while they are already used for many purposes, they aren't yet used as much as they could. Moving contacts (People files) from a system running Haiku to some foreign file-systems, or uploading them by FTP results in an empty file (which it is) but without any of the xattrs containing the contact data, while moving them around other file-systems mangles them in a non-idempotent way into unusable xattrs. People often copy files without caring about xattrs, and even utilities like `cp` don't preserve them without passing extra arguments. Developers therefore under-use them and avoid them for critical information storage.

While reading physical disks from different systems was unlikely in the past, growing usage of networked file-systems and virtualization platforms, using optimized shared folders like VMware or VirtualBox, increases the issue since they are meant to be used from multiple operating systems. Other higher-level protocols and applications, for example `rsync` and `subversion`, and archive formats like `tar` and `star` also try to support some form of extended attributes.

Digital heritage preservation is also a concern, since files are not only interesting for their content, but also their context, including dates and permissions, when copied over file-systems. Projects seeking to preserve digital creation (UK National Archives, POCOS) can operate at various levels. Some will archive full disk images that include the on-disk file-system structures, containing their vendor-specific metadata. Others will prefer archiving files separately, at the risk of losing some of their properties that might be exposed as EA data.

1.1. Example

A 0-byte file on a BFS partition seen from Haiku could carry the following extended attributes:

```
File:  /boot/home/people/François_Revol
Type    Size  Name             Value
'MIMS'    21  "BEOS:TYPE"      "application/x-person"
STRING    14  "META:email"     "revol@free.fr"
STRING     8  "IM:status"      "Offline"
STRING    23  "META:url"       "http://revolf.free.fr/"
RAW       20  "_trk/pinfo_le"  00 BA E3 EC A7 09…
```

After copying this file to an NTFS partition, rebooting to Windows to copy the file to a Samba share running on GNU/Linux (with the xattr support enabled), then rebooting to Haiku to read it back from the ext3 filesystem, the extended attributes might become:

```
File:  /unnamed_ext3/home/revol/François_Revol
Type    Size  Name             Value
RAW      264  "linux.user.DosStreams"
      05 00 00 00 00 00 00…   '................'
      00…-42 45 4f 53 5f…     '........BEOS_TYP'
      45 00 53 4d 49 4d 61…   'E.SMIMapplicatio'…
```

2. Proposed Solution

Instead of having each vendor define its own reserved namespace prefix and mangling scheme for others to implement, we define a common prefix that all vendors should recognize, arbitrarily named `uxa` for Unified eXtended Attributes, and map each vendor's native namespace below it. Vendors then only specify the mangling scheme of the common unified namespace itself into and from their native EA system. This allows each platform to maintain a canonical representation of the xattrs without requiring any knowledge of foreign EA mappings. This uxa namespace does not try to map extended attribute semantics between platforms, but instead focuses on ensuring correct preservation of the original form, and leaves the interpretation of foreign data to higher layers, only providing a generic namespace that encompasses all others, and an idempotent mangling scheme.

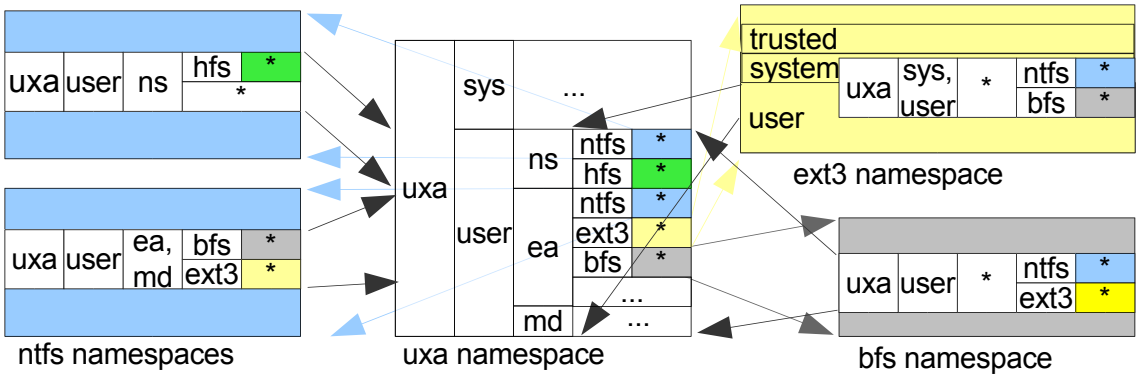

FIG. 1. UXA namespace, the tao of xattr namespaces.

2.1. Namespace Hierarchy

We reserve part of each native EA namespace with the unlikely prefix `uxa`. for Unified eXtended Attributes, to map the uxa namespace and subdivide it further, while keeping the native names unchanged. Foreign EAs would then appear in a branch of the uxa namespace. When copied from a native filesystem to a foreign one, the mapping is reversed: the remaining part of the native namespace appears in the designated branch of the uxa namespace, and existing EAs copied to the branch corresponding to the foreign filesystem are extracted from the uxa namespace. A shortened unix shell pattern-like representation would be:

`uxa.{sys|user}.{ea|ns|md}.{bfs|ntfs|posix|sun|*}.*`

The proposed hierarchy for the uxa namespace follows, using reversed-DNS like notation, though shortened to minimize size and performance penalty on file copying, which shouldn't require special encoding, US-ASCII being sufficient for naming the specified levels. The root level defines the uxa namespace itself as being reserved in native namespaces. The second level separates user-accessible EAs from kernel-only names, though the security implications would likely warrant not using it, due to discrepancies in semantic interpretation of the security restrictions that should apply to them. The third level indicates if the EA originates from a named stream or real EA, since some platforms support both. Other forms of metadata (`md`) are accounted for, to handle mapping POSIX `atime` for example. The last designated level names the platform the EA originates from, and indicates the corresponding mangling scheme to use. The next level maps native namespaces from vendors, and assumes UTF-8 encoding (which includes US-ASCII). Vendor-specific mappings should ensure preservation of the original name, possibly through percent-escaping as with Uniform Resource Locators. Such a candidate is NTFS which disallows some reserved characters like ":" in the named stream designations.

The uxa namespace can be mapped partially multiple times, for example a filesystem supporting both EAs and named streams would map the `uxa.*.ns.*` and `uxa.*.{ea|md}.*` in the respective namespaces. Likewise for user-accessible and restricted namespaces.

In order to better abstract extended attribute mechanisms, we use a model of the traditional OSI network layering (Zimmerman, 1980), and separate the transport and presentation layers. For performance reasons, we propose to split the implementation of the uxa namespace support in Operating Systems. The basic remapping scheme would be implemented in file system layers, typically foreign filesystem kernel modules, while native filesystems will likely not have to mangle the system's native API view of the xattrs, and as such will not suffer any performance loss. The approach taken ensures that foreign file-system implementations only need to care about their respective vendor-defined mangling scheme, since they provide a canonical view to the VFS layer of the OS using the native mangling of the uxa namespace. For example, the "befs.ko" Linux module implementing BFS support would mangle Haiku-specific xattrs as user.uxa.user.ea.bfs.* as specified by Haiku developers and demangle the uxa.user.ea.posix.* BFS EA data as Linux-native user.* xattrs. The BFS mangling scheme will need to encode the type field in some way, either as part of the mangled name, or as part of the value, but this is left to the Haiku developers. In the case of NTFS, the NTFS-3g library could provide the direct uxa namespace canonical representation, and each OS using it would then mangle it back to their native namespace, inside the FUSE kernel modules, for example.

2.1. Higher Level View

Higher level views would be available through either a custom library exposing more semantics, or a reimplemented libxattr compatible with the Linux API for faster portability. This library could try to automatically convert foreign attributes to standardized views, for example the FreeDesktop.org set which includes the Simple Dublin Core Metadata Element Set (FreeDesktop.org 1 & 2). Other possibilities could include a Java API extension, building on (JSR 203). Conversion tools can also be written to help users migrate their (meta)data. The proposed scheme could also finally be taken as a canonical representation for use in backing-store systems to replace platform-specific files that clutter directories like `.DS_Store` or the FreeDesktop.org proposed solution.

EAs requiring conversion include the MIME type of the file, defined as a value of type "MIMS" and name "BEOS:TYPE" in Haiku, a string named "user.mime_type" on FreeDesktop-compliant platforms, or a list of Uniform Type Identifiers (UTI) on Mac OS X.

Other metadata of interest is the originating URI of a downloaded file. Mac OS X stores this as a binary structure (property list) of name "com.apple.metadata:kMDItemWhereFroms". BeOS and Haiku would instead use a string typed "CSTR" of name "META:url". FreeDesktop specifies it as "user.xdg.origin.url".

3. Shortcomings

The uxa namespace mapping doesn't consider file-systems with limited storage capabilities, and assumes small enough attributes and names, as a best-effort solution. The uxa namespace design tries to minimize the length of the name prefixes. As noted, ext2, 3 and 4 file-systems only allow a single disk block per inode for xattr storage, including names. Other filesystems like XFS allow larger xattrs, 64kB per xattr, and up to 255 bytes for the xattr names in this case. BFS doesn't impose any limitation of xattr size but is able to store small ones in the "small_data" section of the inode. It is expected that system administrators will be made aware of the problem and account for xattr size limits when choosing a file-system for archiving purposes. Likewise, file-system designers are expected to include fair provision for EA data size in their decisions.

A fall-back strategy should probably use backing files, as with file-systems missing extended attribute support. Some operating systems support extended attributes only on regular files, not symlinks or directories. ACL mappings aren't accounted for and are already subject to much discussion in the literature. The low-level mapping only focuses on preserving the xattrs across file-systems and operating systems, and does not consider their meaning or even their content. Software accessing xattrs on files without further processing will not likely have any use for

those imported from other operating systems, but will at least be able to keep them when transferring data.

Higher-level support is still subject to broad discussion about which semantic level to achieve. Also, the proposed higher-level xattr transformations do not consider the possible lack of synchronization between the original metadata and their converted form attached to the same file, i.e., a file could end up with both a BEOS:TYPE and user.mime_type xattr with different values.

A centralized registry, like the one used for MIME types (IANA), should be designated to attribute vendor names into the uxa namespace, and mangling schemes should be publicly disclosed and agreed upon, through RFCs for example, to be successfully usable. Finally it is also unlikely that proprietary software vendors will ever conform to such a scheme without a strong standardization effort, but implementation by most OpenSource projects could help the process.

4. Conclusion

The purpose of this early work was to raise the concern about extended attributes interoperability, propose a possible solution using an idempotent mangling scheme of a common namespace allowing further processing at higher layers, and foster discussion between involved parties, possibly leading to a standardized document like an IETF Request For Comment, with vendors defining the global namespace mapping to their own filesystems and protocols. Both OpenSource and proprietary projects would benefit from standardized extended attribute interoperability schemes allowing platforms to maintain each other's metadata across file transfers, making them more useful. Additionally, applications would benefit from having a standardized view on existing foreign extended attributes.

References

attr(5) mainpage. http://linux.die.net/man/5/attr

FreeDesktop.org 1: Common Extended Attributes. http://www.freedesktop.org/wiki/CommonExtendedAttributes

FreeDesktop.org 2: Shared File MetaData. http://www.freedesktop.org/wiki/Specifications/shared-filemetadata-spec

French, S. M., Kleikamp, D.K. and Tso, T.Y.T. (2008). Method and apparatus for emulating alternate data streams across heterogeneous file systems, 03 2008. (IBM) (US patent 2008/0065698 A1).

Giampaolo. D. (1999). Practical file system design with the BE file system. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA.

Haiku Operating System. http://www.haiku-os.org/

IANA (Internet Assigned Numbers Association). http://www.iana.org/

JSR 203. http://www.jcp.org/en/jsr/detail?id=203

Leung, A.W., Shao, M., Bisson, T., Pasupathy, S. and Miller, E.L. (2008). High-performance metadata indexing and search in petascale data storage systems. Journal of Physics: Conference Series, 125:012069 (5pp).

Möller, K. and Handschuh, S. (June 2007). Towards a light-weight semantic desktop. In Proceedings of the Semantic Desktop Design Workshop (SemDeskDesign 2007) at ESWC2007, Innsbruck, Austria, Innsbruck, Austria.

Morris, J. (2009). Adding extended attribute support to NFS. http://namei.org/presentations/linuxcon09_nfsv3xattrs.pdf

NTFS-3g extended attributes. http://www.tuxera.com/community/ntfs-3g-advanced/extended-attributes/

POCOS (Preservation of Complex Objects). http://pocos.org/

Posix 1003.1e / 1003.2c Draft Standard 17 (withdrawn). http://www.suse.de/~agruen/acl/posix/posix.html

Tridgell, A. (2005). Wine/samba. http://www.samba.org/samba/ftp/slides/tridge_wineconf05.pdf (Wineconf).

UK National Archives. http://www.nationalarchives.gov.uk/information-management/projects-and-work/digital-preservation.htm

Zimmerman, H. (1980). Osi reference model - the iso model of architecture for open systems interconnection. IEEE Transactions on Communications, (28).