NAME

acl — Access Control Lists

DESCRIPTION

This manual page describes POSIX Access Control Lists, which are used to define more fine-grained discretionary access rights for files and directories.

ACL TYPES

Every object can be thought of as having associated with it an ACL that governs the discretionary access to that object; this ACL is referred to as an access ACL. In addition, a directory may have an associated ACL that governs the initial access ACL for objects created within that directory; this ACL is referred to as a default ACL.

ACL ENTRIES

An ACL consists of a set of ACL entries. An ACL entry specifies the access permissions on the associated object for an individual user or a group of users as a combination of read, write and search/execute permissions.

An ACL entry contains an entry tag type, an optional entry tag qualifier, and a set of permissions. We use the term qualifier to denote the entry tag qualifier of an ACL entry.

The qualifier denotes the identifier of a user or a group, for entries with tag types of ACL_USER or ACL_GROUP, respectively. Entries with tag types other than ACL_USER or ACL_GROUP have no defined qualifiers.

The following entry tag types are defined:

ACL_USER_OBJ	The ACL_USER_OBJ entry denotes access rights for the file owner.
ACL_USER	ACL_USER entries denote access rights for users identified by the entry's qualifier.
ACL_GROUP_OBJ	The ACL_GROUP_OBJ entry denotes access rights for the file group.
ACL_GROUP	ACL_GROUP entries denote access rights for groups identified by the entry's qualifier.
ACL_MASK	The ACL_MASK entry denotes the maximum access rights that can be granted by entries of type ACL_USER, ACL_GROUP_OBJ, or ACL_GROUP.
ACL_OTHER	The ACL_OTHER entry denotes access rights for processes that do not match any other entry in the ACL.

When an access check is performed, the ACL_USER_OBJ and ACL_USER entries are tested against the effective user ID. The effective group ID, as well as all supplementary group IDs are tested against the ACL_GROUP_OBJ and ACL_GROUP entries.

VALID ACLs

A valid ACL contains exactly one entry with each of the ACL_USER_OBJ, ACL_GROUP_OBJ, and ACL_OTHER tag types. Entries with ACL_USER and ACL_GROUP tag types may appear zero or more times in an ACL. An ACL that contains entries of ACL_USER or ACL_GROUP tag types must contain exactly one entry of the ACL_MASK tag type. If an ACL contains no entries of ACL_USER or ACL_GROUP tag types, the ACL_MASK entry is optional.

All user ID qualifiers must be unique among all entries of ACL_USER tag type, and all group IDs must be unique among all entries of ACL_GROUP tag type.

The **acl_get_file**() function returns an ACL with zero ACL entries as the default ACL of a directory, if the directory is not associated with a default ACL. The **acl_set_file**() function also accepts an ACL with zero ACL entries as a valid default ACL for directories, denoting that the directory shall not be associated with a default ACL. This is equivalent to using the **acl_delete_def_file**() function.

CORRESPONDENCE BETWEEN ACL ENTRIES AND FILE PERMISSION BITS

The permissions defined by ACLs are a superset of the permissions specified by the file permission bits.

There is a correspondence between the file owner, group, and other permissions and specific ACL entries: the owner permissions correspond to the permissions of the ACL_USER_OBJ entry. If the ACL has an ACL_MASK entry, the group permissions correspond to the permissions of the ACL_MASK entry. Otherwise, if the ACL has no ACL_MASK entry, the group permissions correspond to the permissions of the ACL_GROUP_OBJ entry. The other permissions correspond to the permissions of the ACL_OTHER_OBJ entry.

The file owner, group, and other permissions always match the permissions of the corresponding ACL entry. Modification of the file permission bits results in the modification of the associated ACL entries, and modification of these ACL entries results in the modification of the file permission bits.

OBJECT CREATION AND DEFAULT ACLs

The access ACL of a file object is initialized when the object is created with any of the **creat**(), **mkdir**(), **mknod**(), **mkfifo**(), or **open**() functions. If a default ACL is associated with a directory, the *mode* parameter to the functions creating file objects and the default ACL of the directory are used to determine the ACL of the new object:

1. The new object inherits the default ACL of the containing directory as its access ACL.
2. The access ACL entries corresponding to the file permission bits are modified so that they contain no permissions that are not contained in the permissions specified by the *mode* parameter.

If no default ACL is associated with a directory, the *mode* parameter to the functions creating file objects and the file creation mask (see umask(2)) are used to determine the ACL of the new object:

1. The new object is assigned an access ACL containing entries of tag types ACL_USER_OBJ, ACL_GROUP_OBJ, and ACL_OTHER. The permissions of these entries are set to the permissions specified by the file creation mask.
2. The access ACL entries corresponding to the file permission bits are modified so that they contain no permissions that are not contained in the permissions specified by the *mode* parameter.

ACCESS CHECK ALGORITHM

A process may request read, write, or execute/search access to a file object protected by an ACL. The access check algorithm determines whether access to the object will be granted.

1. **If** the effective user ID of the process matches the user ID of the file object owner, **then**

   if the ACL_USER_OBJ entry contains the requested permissions, access is granted,

   else access is denied.

2. **else if** the effective user ID of the process matches the qualifier of any entry of type ACL_USER, **then**

   if the matching ACL_USER entry and the ACL_MASK entry contain the requested permissions, access is granted,

   else access is denied.

3. **else if** the effective group ID or any of the supplementary group IDs of the process match the file group or the qualifier of any entry of type ACL_GROUP, **then**

   if the ACL contains an ACL_MASK entry, **then**

   if the ACL_MASK entry and any of the matching ACL_GROUP_OBJ or ACL_GROUP entries contain the requested permissions, access is granted,

   else access is denied.

   else (note that there can be no ACL_GROUP entries without an ACL_MASK entry)

   if the ACL_GROUP_OBJ entry contains the requested permissions, access is granted,

   else access is denied.

4. **else if** the ACL_OTHER entry contains the requested permissions, access is granted.
5. **else** access is denied.

ACL TEXT FORMS

A long and a short text form for representing ACLs is defined. In both forms, ACL entries are represented as three colon separated fields: an ACL entry tag type, an ACL entry qualifier, and the discretionary access permissions. The first field contains one of the following entry tag type keywords:

`user` A `user` ACL entry specifies the access granted to either the file owner (entry tag type ACL_USER_OBJ) or a specified user (entry tag type ACL_USER).

`group` A `group` ACL entry specifies the access granted to either the file group (entry tag type ACL_GROUP_OBJ) or a specified group (entry tag type ACL_GROUP).

`mask` A `mask` ACL entry specifies the maximum access which can be granted by any ACL entry except the `user` entry for the file owner and the `other` entry (entry tag type ACL_MASK).

`other` An other ACL entry specifies the access granted to any process that does not match any `user` or `group` ACL entries (entry tag type ACL_OTHER).

The second field contains the user or group identifier of the user or group associated with the ACL entry for entries of entry tag type ACL_USER or ACL_GROUP, and is empty for all other entries. A user identifier can be a user name or a user ID number in decimal form. A group identifier can be a group name or a group ID number in decimal form.

The third field contains the discretionary access permissions. The read, write and search/execute permissions are represented by the `r`, `w`, and `x` characters, in this order. Each of these characters is replaced by the `-` character to denote that a permission is absent in the ACL entry. When converting from the text form to the internal representation, permissions that are absent need not be specified.

White space is permitted at the beginning and end of each ACL entry, and immediately before and after a field separator (the colon character).

LONG TEXT FORM

The long text form contains one ACL entry per line. In addition, a number sign (#) may start a comment that extends until the end of the line. If an ACL_USER, ACL_GROUP_OBJ or ACL_GROUP ACL entry contains permissions that are not also contained in the ACL_MASK entry, the entry is followed by a number sign, the string “effective:”, and the effective access permissions defined by that entry. This is an example of the long text form:

```
user::rw-
user:lisa:rw- #effective:r--
group::r--
group:toolies:rw- #effective:r--
mask::r--
other::r--
```

SHORT TEXT FORM

The short text form is a sequence of ACL entries separated by commas, and is used for input. Comments are not supported. Entry tag type keywords may either appear in their full unabbreviated form, or in their single letter abbreviated form. The abbreviation for `user` is `u`, the abbreviation for `group` is `g`, the abbreviation for `mask` is `m`, and the abbreviation for `other` is `o`. The permissions may contain at most one each of the following characters in any order: `r`, `w`, `x`. These are examples of the short text form:

```
u::rw-,u:lisa:rw-,g::r--,g:toolies:rw-,m::r--,o::r--
g:toolies:rw,u:lisa:rw,u::wr,g::r,o::r,m::r
```

RATIONALE

IEEE 1003.1e draft 17 defines Access Control Lists that include entries of tag type ACL_MASK, and defines a mapping between file permission bits that is not constant. The standard working group defined this relatively complex interface in order to ensure that applications that are compliant with IEEE 1003.1 ("POSIX.1") will still function as expected on systems with ACLs. The IEEE 1003.1e draft 17 contains the rationale for choosing this interface in section B.23.

CHANGES TO THE FILE UTILITIES

On a system that supports ACLs, the file utilities ls(1), cp(1), and `mv`(1) change their behavior in the following way:

- For files that have a default ACL or an access ACL that contains more than the three required ACL entries, the `ls`(1) utility in the long form produced by **`ls -l`** displays a plus sign (`+`) after the permission string.
- If the **`-p`** flag is specified, the `cp`(1) utility also preserves ACLs. If this is not possible, a warning is produced.
- The `mv`(1) utility always preserves ACLs. If this is not possible, a warning is produced.

The effect of the `chmod`(1) utility, and of the `chmod`(2) system call, on the access ACL is described in CORRESPONDENCE BETWEEN ACL ENTRIES AND FILE PERMISSION BITS.

STANDARDS

The IEEE 1003.1e draft 17 ("POSIX.1e") document describes several security extensions to the IEEE 1003.1 standard. While the work on 1003.1e has been abandoned, many UNIX style systems implement parts of POSIX.1e draft 17, or of earlier drafts.

Linux Access Control Lists implement the full set of functions and utilities defined for Access Control Lists in POSIX.1e, and several extensions. The implementation is fully compliant with POSIX.1e draft 17; extensions are marked as such. The Access Control List manipulation functions are defined in the ACL library (libacl, -lacl). The POSIX compliant interfaces are declared in the `<sys/acl.h>` header. Linux-specific extensions to these functions are declared in the `<acl/libacl.h>` header.

SEE ALSO

chmod(1), creat(2), getfacl(1), ls(1), mkdir(2), mkfifo(2), mknod(2), open(2), setfacl(1), stat(2), `umask`(1)

POSIX 1003.1e DRAFT 17

http://www.guug.de/~winni/posix.1e/download.html -P "

POSIX 1003.1e FUNCTIONS BY CATEGORY

ACL storage management

acl_dup(3), acl_free(3), `acl_init`(3)

ACL entry manipulation

acl_copy_entry(3), acl_create_entry(3), acl_delete_entry(3), acl_get_entry(3), `acl_valid`(3)

acl_add_perm(3), acl_calc_mask(3), acl_clear_perms(3), acl_delete_perm(3), acl_get_permset(3), `acl_set_permset`(3)

acl_get_qualifier(3), acl_get_tag_type(3), acl_set_qualifier(3), `acl_set_tag_type`(3)

ACL manipulation on an object

acl_delete_def_file(3), acl_get_fd(3), acl_get_file(3), acl_set_fd(3), `acl_set_file`(3)

ACL format translation

acl_copy_entry(3), acl_copy_ext(3), acl_from_text(3), acl_to_text(3), `acl_size`(3)

POSIX 1003.1e FUNCTIONS BY AVAILABILITY

The first group of functions is supported on most systems with POSIX-like access control lists, while the second group is supported on fewer systems. For applications that will be ported the second group is best avoided.

acl_delete_def_file(3), acl_dup(3), acl_free(3), acl_from_text(3), acl_get_fd(3), acl_get_file(3), acl_init(3), acl_set_fd(3), acl_set_file(3), acl_to_text(3), `acl_valid`(3)

acl_add_perm(3), acl_calc_mask(3), acl_clear_perms(3), acl_copy_entry(3), acl_copy_ext(3), acl_copy_int(3), acl_create_entry(3), acl_delete_entry(3), acl_delete_perm(3), acl_get_entry(3), acl_get_permset(3), acl_get_qualifier(3), acl_get_tag_type(3), acl_set_permset(3), acl_set_qualifier(3), acl_set_tag_type(3), `acl_size`(3)

LINUX EXTENSIONS

These non-portable extensions are available on Linux systems.

acl_check(3), acl_cmp(3), acl_entries(3), acl_equiv_mode(3), acl_error(3), acl_extended_fd(3), acl_extended_file(3), acl_extended_file_nofollow(3), acl_from_mode(3), acl_get_perm(3), `acl_to_any_text`(3)

AUTHOR

Andreas Gruenbacher, <a.gruenbacher@bestbits.at>