NAME

attr − extended attributes on XFS filesystem objects

SYNOPSIS

attr [**−LRq**] **−s attrname** [**−V attrvalue**] **pathname**

attr [**−LRq**] **−g attrname pathname**

attr [**−LRq**] **−r attrname pathname**

OVERVIEW

Extended attributes implement the ability for a user to attach name:value pairs to objects within the XFS filesystem.

They could be used to store meta-information about the file. For example "character-set=kanji" could tell a document browser to use the Kanji character set when displaying that document and "thumbnail=..." could provide a reduced resolution overview of a high resolution graphic image.

This document describes the *attr* command, which is mostly compatible with the IRIX command of the same name. It is thus aimed specifically at users of the XFS filesystem - for filesystem independent extended attribute manipulation, consult the *getfattr*(1) and *setfattr*(1) documentation.

In the XFS filesystem, the *names* can be up to 256 bytes in length, terminated by the first 0 byte. The intent is that they be printable ASCII (or other character set) names for the attribute. The *values* can be up to 64KB of arbitrary binary data.

Attributes can be attached to all types of XFS inodes: regular files, directories, symbolic links, device nodes, etc.

XFS uses 2 disjoint attribute name spaces associated with every filesystem object. They are the **root** and **user** address spaces. The **root** address space is accessable only to the superuser, and then only by specifying a flag argument to the function call. Other users will not see or be able to modify attributes in the **root** address space. The **user** address space is protected by the normal file permissions mechanism, so the owner of the file can decide who is able to see and/or modify the value of attributes on any particular file.

DESCRIPTION

The *attr* utility allows the manipulation of extended attributes associated with filesystem objects from within shell scripts.

There are four main operations that *attr* can perform:

GET The **−g attrname** option tells *attr* to search the named object and print (to ***stdout***) the value associated with that attribute name. With the **−q** flag, ***stdout*** will be exactly and only the value of the attribute, suitable for storage directly into a file or processing via a piped command.

REMOVE

The **−r attrname** option tells *attr* to remove an attribute with the given name from the object if the attribute exists. There is no output on sucessful completion.

SET/CREATE

The **−s attrname** option tells *attr* to set the named attribute of the object to the value read from ***stdin***. If an attribute with that name already exists, its value will be replaced with this one. If an attribute with that name does not already exist, one will be created with this value. With the **−V attrvalue** flag, the attribute will be set to have a value of **attrvalue** and ***stdin*** will not be read. With the **−q** flag, ***stdout*** will not be used. Without the **−q** flag, a message showing the attribute name and the entire value will be printed.

When the **−L** option is given and the named object is a symbolic link, operate on the attributes of the object referenced by the symbolic link. Without this option, operate on the attributes of the symbolic link itself.

When the **−R** option is given and the process has appropriate privileges, operate in the *root* attribute namespace rather that the *USER* attribute namespace.

When the **−q** option is given *attr* will try to keep quiet. It will output error messages (to ***stderr***) but will

not print status messages (to ***stdout***).

NOTES

The standard file interchange/archive programs *tar*(1), and *cpio*(1) will not archive or restore extended attributes, while the *xfsdump*(8) program will.

CAVEATS

The list option present in the IRIX version of this command is not supported. *getfattr* provides a mechanism to retrieve all of the attribute names.

SEE ALSO

getfattr(1), setfattr(1), attr_get(3), attr_set(3), attr_multi(3), attr_remove(3), attr(5), and xfsdump(8).

NAME

getfattr − get extended attributes of filesystem objects

SYNOPSIS

getfattr [**−hRLP**] **−n name** [**−e en**] **pathname**...
getfattr [**−hRLP**] **−d** [**−e en**] [**−m pattern**] **pathname**...

DESCRIPTION

For each file, **getfattr** displays the file name, and the set of extended attribute names (and optionally values) which are associated with that file.

The output format of **getfattr −d** is as follows:

```
1:  # file: somedir/
2:  user.name0="value0"
3:  user.name1="value1"
4:  user.name2="value2"
5:  ...
```

Line 1 identifies the file name for which the following lines are being reported. The remaining lines (lines 2 to 4 above) show the *name* and *value* pairs associated with the specified file.

OPTIONS

−n *name*, **−−name**=*name*
Dump the value of the named extended attribute extended attribute.

−d, **−−dump**
Dump the values of all extended attributes associated with *pathname*.

−e *en*, **−−encoding**=*en*
Encode values after retrieving them. Valid values of *en* are "text", "hex", and "base64". Values encoded as text strings are enclosed in double quotes ("), while strings encoded as hexidecimal and base64 are prefixed with 0x and 0s, respectively.

−h, **−−no-dereference**
Do not follow symlinks. If *pathname* is a symbolic link, the symbolic link itself is examined, rather than the file the link refers to.

−m *pattern*, **−−match**=*pattern*
Only include attributes with names matching the regular expression *pattern*. The default value for *pattern* is "^user\\.", **−m** which includes all the attributes in the user namespace. Refer to **attr**(5) for a more detailed discussion on namespaces.

−−absolute-names
Do not strip leading slash characters (’/’). The default behaviour is to strip leading slash characters.

−−only-values
Dump out the extended attribute value(s) only.

−R, **−−recursive**
List the attributes of all files and directories recursively.

−L, **−−logical**
Logical walk, follow symbolic links. The default behaviour is to follow symbolic link arguments, and to skip symbolic links encountered in subdirectories.

−P, **−−physical**
Physical walk, skip all symbolic links. This also skips symbolic link arguments.

−−version
Print the version of **getfattr** and exit.

−−help
Print help explaining the command line options.

−− End of command line options. All remaining parameters are interpreted as file names, even if they start with a dash character.

AUTHOR

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>.

Please send your bug reports or comments to these addresses.

SEE ALSO

setfattr(1), and attr(5).

NAME

setfattr − set extended attributes of filesystem objects

SYNOPSIS

setfattr [**−h**] **−n name** [**−v value**] **pathname**...
setfattr [**−h**] **−x name pathname**...
setfattr [**−h**] **−−restore=file**

DESCRIPTION

The **setfattr** command associates a new *value* with an extended attribute *name* for each specified file.

OPTIONS

−n *name*, **−−name**=*name*
Specifies the name of the extended attribute to set.

−v *value*, **−−value**=*value*
Specifies the new value for the extended attribute.

−x *name*, **−−remove**=*name*
Remove the named extended attribute entirely.

−h, **−−no-dereference**
Do not follow symlinks. If *pathname* is a symbolic link, it is not followed, but is instead itself the inode being modified.

−−restore=*file*
Restores extended attributes from file. The file must be in the format generated by the **getfattr** command with the **−−dump** option. If a dash (−) is given as the file name, **setfattr** reads from standard input.

−−version
Print the version of **setfattr** and exit.

−−help
Print help explaining the command line options.

−− End of command line options. All remaining parameters are interpreted as file names, even if they start with a dash character.

AUTHOR

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>.

Please send your bug reports or comments to these addresses.

SEE ALSO

getfattr(1), and attr(5).

NAME

getxattr, lgetxattr, fgetxattr − retrieve an extended attribute value

SYNOPSIS

```
#include <sys/types.h>
#include <attr/xattr.h>

ssize_t getxattr (const char *path, const char *name,
                  void *value, size_t size);
ssize_t lgetxattr (const char *path, const char *name,
                  void *value, size_t size);
ssize_t fgetxattr (int filedes, const char *name,
                  void *value, size_t size);
```

DESCRIPTION

Extended attributes are *name*:*value* pairs associated with inodes (files, directories, symlinks, etc). They are extensions to the normal attributes which are associated with all inodes in the system (i.e. the **stat**(2) data). A complete overview of extended attributes concepts can be found in **attr**(5).

getxattr retrieves the *value* of the extended attribute identified by *name* and associated with the given *path* in the filesystem. The length of the attribute *value* is returned.

lgetxattr is identical to **getxattr**, except in the case of a symbolic link, where the link itself is interrogated, not the file that it refers to.

fgetxattr is identical to **getxattr**, only the open file pointed to by *filedes* (as returned by **open**(2)) is interrogated in place of *path*.

An extended attribute *name* is a simple NULL-terminated string. The name includes a namespace prefix − there may be several, disjoint namespaces associated with an individual inode. The value of an extended attribute is a chunk of arbitrary textual or binary data of specified length.

An empty buffer of *size* zero can be passed into these calls to return the current size of the named extended attribute, which can be used to estimate the size of a buffer which is sufficiently large to hold the value associated with the extended attribute.

The interface is designed to allow guessing of initial buffer sizes, and to enlarge buffers when the return value indicates that the buffer provided was too small.

RETURN VALUE

On success, a positive number is returned indicating the size of the extended attribute value. On failure, −1 is returned and *errno* is set appropriately.

If the named attribute does not exist, or the process has no access to this attribute, *errno* is set to ENOATTR.

If the *size* of the *value* buffer is too small to hold the result, *errno* is set to ERANGE.

If extended attributes are not supported by the filesystem, or are disabled, *errno* is set to ENOTSUP.

The errors documented for the **stat**(2) system call are also applicable here.

AUTHORS

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>. Please send any bug reports or comments to these addresses.

SEE ALSO

getfattr(1), **setfattr**(1), **open**(2), **stat**(2), **setxattr**(2), **listxattr**(2), **removexattr**(2), and **attr**(5).

NAME

listxattr, llistxattr, flistxattr − list extended attribute names

SYNOPSIS

```
#include <sys/types.h>
#include <attr/xattr.h>

ssize_t listxattr (const char *path,
                   char *list, size_t size);
ssize_t llistxattr (const char *path,
                   char *list, size_t size);
ssize_t flistxattr (int filedes,
                   char *list, size_t size);
```

DESCRIPTION

Extended attributes are name:value pairs associated with inodes (files, directories, symlinks, etc). They are extensions to the normal attributes which are associated with all inodes in the system (i.e. the **stat**(2) data). A complete overview of extended attributes concepts can be found in **attr**(5).

listxattr retrieves the *list* of extended attribute names associated with the given *path* in the filesystem. The list is the set of (NULL-terminated) names, one after the other. Names of extended attributes to which the calling process does not have access may be omitted from the list. The length of the attribute name *list* is returned.

llistxattr is identical to **listxattr**, except in the case of a symbolic link, where the list of names of extended attributes associated with the link itself is retrieved, not the file that it refers to.

flistxattr is identical to **listxattr**, only the open file pointed to by *filedes* (as returned by **open**(2)) is interrogated in place of *path*.

A single extended attribute *name* is a simple NULL-terminated string. The name includes a namespace prefix − there may be several, disjoint namespaces associated with an individual inode.

An empty buffer of *size* zero can be passed into these calls to return the current size of the list of extended attribute names, which can be used to estimate the size of a buffer which is sufficiently large to hold the list of names.

EXAMPLES

The *list* of names is returned as an unordered array of NULL-terminated character strings (attribute names are separated by NULL characters), like this:

```
user.name1\0system.name1\0user.name2\0
```

Filesystems like ext2, ext3 and XFS which implement POSIX ACLs using extended attributes, might return a *list* like this:

```
system.posix_acl_access\0system.posix_acl_default\0
```

RETURN VALUE

On success, a positive number is returned indicating the size of the extended attribute name list. On failure, −1 is returned and *errno* is set appropriately.

If the *size* of the *list* buffer is too small to hold the result, *errno* is set to ERANGE.

If extended attributes are not supported by the filesystem, or are disabled, *errno* is set to ENOTSUP.

The errors documented for the **stat**(2) system call are also applicable here.

AUTHORS

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>. Please send any bug reports or comments to these addresses.

SEE ALSO

getfattr(1), **setfattr**(1), **open**(2), **stat**(2), **getxattr**(2), **setxattr**(2), **removexattr**(2), and **attr**(5).

NAME

removexattr, lremovexattr, fremovexattr − remove an extended attribute

SYNOPSIS

```
#include <sys/types.h>
#include <attr/xattr.h>

int removexattr (const char *path, const char *name);
int lremovexattr (const char *path, const char *name);
int fremovexattr (int filedes, const char *name);
```

DESCRIPTION

Extended attributes are *name*:value pairs associated with inodes (files, directories, symlinks, etc). They are extensions to the normal attributes which are associated with all inodes in the system (i.e. the **stat**(2) data). A complete overview of extended attributes concepts can be found in **attr**(5).

removexattr removes the extended attribute identified by *name* and associated with the given *path* in the filesystem.

lremovexattr is identical to **removexattr**, except in the case of a symbolic link, where the extended attribute is removed from the link itself, not the file that it refers to.

fremovexattr is identical to **removexattr**, only the extended attribute is removed from the open file pointed to by *filedes* (as returned by **open**(2)) in place of *path*.

An extended attribute name is a simple NULL-terminated string. The *name* includes a namespace prefix − there may be several, disjoint namespaces associated with an individual inode.

RETURN VALUE

On success, zero is returned. On failure, −1 is returned and *errno* is set appropriately.

If the named attribute does not exist, *errno* is set to ENOATTR.

If extended attributes are not supported by the filesystem, or are disabled, *errno* is set to ENOTSUP.

The errors documented for the **stat**(2) system call are also applicable here.

AUTHORS

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>. Please send any bug reports or comments to these addresses.

SEE ALSO

getfattr(1), **setfattr**(1), **open**(2), **stat**(2), **setxattr**(2), **getxattr**(2), **listxattr**(2), and **attr**(5).

NAME

setxattr, lsetxattr, fsetxattr − set an extended attribute value

SYNOPSIS

```
#include <sys/types.h>
#include <attr/xattr.h>

int setxattr (const char *path, const char *name,
              const void *value, size_t size, int flags);
int lsetxattr (const char *path, const char *name,
              const void *value, size_t size, int flags);
int fsetxattr (int filedes, const char *name,
              const void *value, size_t size, int flags);
```

DESCRIPTION

Extended attributes are *name*:*value* pairs associated with inodes (files, directories, symlinks, etc). They are extensions to the normal attributes which are associated with all inodes in the system (i.e. the **stat**(2) data). A complete overview of extended attributes concepts can be found in **attr**(5).

setxattr sets the *value* of the extended attribute identified by *name* and associated with the given *path* in the filesystem. The *size* of the *value* must be specified.

lsetxattr is identical to **setxattr**, except in the case of a symbolic link, where the extended attribute is set on the link itself, not the file that it refers to.

fsetxattr is identical to **setxattr**, only the extended attribute is set on the open file pointed to by *filedes* (as returned by **open**(2)) in place of *path*.

An extended attribute name is a simple NULL-terminated string. The *name* includes a namespace prefix − there may be several, disjoint namespaces associated with an individual inode. The *value* of an extended attribute is a chunk of arbitrary textual or binary data of specified length.

The *flags* parameter can be used to refine the semantics of the operation. XATTR_CREATE specifies a pure create, which fails if the named attribute exists already. XATTR_REPLACE specifies a pure replace operation, which fails if the named attribute does not already exist. By default (no flags), the extended attribute will be created if need be, or will simply replace the value if the attribute exists.

RETURN VALUE

On success, zero is returned. On failure, −1 is returned and *errno* is set appropriately.

If XATTR_CREATE is specified, and the attribute exists already, *errno* is set to EEXIST. If XATTR_REPLACE is specified, and the attribute does not exist, *errno* is set to ENOATTR.

If there is insufficient space remaining to store the extended attribute, *errno* is set to either ENOSPC, or EDQUOT if quota enforcement was the cause.

If extended attributes are not supported by the filesystem, or are disabled, *errno* is set to ENOTSUP.

The errors documented for the **stat**(2) system call are also applicable here.

AUTHORS

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>. Please send any bug reports or comments to these addresses.

SEE ALSO

getfattr(1), **setfattr**(1), **open**(2), **stat**(2), **getxattr**(2), **listxattr**(2), **removexattr**(2), and **attr**(5).

NAME

attr_get, attr_getf − get the value of a user attribute of a filesystem object

C SYNOPSIS

#include <attr/attributes.h>

int attr_get (const char ****path*, const char ****attrname***,**
char ****attrvalue*, int *****valuelength***, int** *flags***);**

int attr_getf (int *fd***, const char *****attrname***,**
char *****attrvalue*, int *****valuelength***, int** *flags***);**

DESCRIPTION

The **attr_get** and **attr_getf** functions provide a way to retrieve the value of an attribute.

Path points to a path name for a filesystem object, and *fd* refers to the file descriptor associated with a file. If the attribute *attrname* exists, the value associated with it will be copied into the *attrvalue* buffer. The *valuelength* argument is an input/output argument that on the call to **attr_get** should contain the maximum size of attribute value the process is willing to accept. On return, the *valuelength* will have been modified to show the actual size of the attribute value returned. The *flags* argument can contain the following symbols bitwise OR´ed together:

ATTR_ROOT
: Look for *attrname* in the **root** address space, not in the **user** address space. (limited to use by super-user only)

ATTR_DONTFOLLOW
: Do not follow symbolic links when resolving a *path* on an **attr_get** function call. The default is to follow symbolic links.

attr_get will fail if one or more of the following are true:

[ENOATTR] The attribute name given is not associated with the indicated filesystem object.

[E2BIG] The value of the given attribute is too large to fit into the buffer. The integer that the *valuelength* argument points to has been modified to show the actual number of bytes that would be required to store the value of that attribute.

[ENOENT] The named file does not exist.

[EPERM] The effective user ID does not match the owner of the file and the effective user ID is not super-user.

[ENOTDIR] A component of the path prefix is not a directory.

[EACCES] Search permission is denied on a component of the path prefix.

[EINVAL] A bit was set in the *flag* argument that is not defined for this system call.

[EFAULT] *Path, attrname, attrvalue,* or *valuelength* points outside the allocated address space of the process.

[ELOOP] A path name lookup involved too many symbolic links.

[ENAMETOOLONG] The length of *path* exceeds {*MAXPATHLEN*}, or a pathname component is longer than {*MAXNAMELEN*}.

attr_getf will fail if:

[ENOATTR] The attribute name given is not associated with the indicated filesystem object.

[E2BIG] The value of the given attribute is too large to fit into the buffer. The integer that the *valuelength* argument points to has been modified to show the actual numnber of bytes that would be required to store the value of that attribute.

[EINVAL] A bit was set in the *flag* argument that is not defined for this system call, or *fd* refers to a socket, not a file.

[EFAULT] *Attrname, attrvalue,* or *valuelength* points outside the allocated address space of the process.

[EBADF] *Fd* does not refer to a valid descriptor.

DIAGNOSTICS

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

SEE ALSO

attr(1), **attr_multi**(3), **attr_remove**(3), and **attr_set**(3).

NAME

attr_multi, attr_multif − manipulate multiple user attributes on a filesystem object at once

C SYNOPSIS

#include <attr/attributes.h>

int attr_multi (const char **path*, **attr_multiop_t ****oplist*,
int *count*, **int** *flags*);**

int attr_multif (int *fd*, **attr_multiop_t ****oplist*,**
int *count*, **int** *flags*);**

DESCRIPTION

The **attr_multi** and **attr_multif** functions provide a way to operate on multiple attributes of a filesystem object at once.

Path points to a path name for a filesystem object, and *fd* refers to the file descriptor associated with a file. The *oplist* is an array of ***attr_multiop_t*** structures. Each element in that array describes a single attribute operation and provides all the information required to carry out that operation and to check for success or failure of that operation. *Count* tells how many elements are in the *oplist* array.

The contents of an ***attr_multiop_t*** structure include the following members:

int am_opcode; /* which operation to perform (see below) */
int am_error; /* [out arg] result of this sub-op (an errno) */
char *am_attrname; /* attribute name to work with */
char *am_attrvalue; /* [in/out arg] attribute value (raw bytes) */
int am_length; /* [in/out arg] length of value */
int am_flags; /* flags (bit-wise OR of #defines below) */

The *am_opcode* field defines how the remaining fields are to be interpreted and can take on one of the following values:

ATTR_OP_GET /* return the indicated attr's value */
ATTR_OP_SET /* set/create the indicated attr/value pair */
ATTR_OP_REMOVE /* remove the indicated attr */

The *am_error* field will contain the appropriate error result code if that sub-operation fails. The result codes for a given sub-operation are a subset of the result codes that are possible from the corresponding single-attribute function call. For example, the result code possible from an ***ATTR_OP_GET*** sub-operation are a subset of those that can be returned from an **attr_get** function call.

The *am_attrname* field is a pointer to a NULL terminated string giving the attribute name that the sub-operation should operate on.

The *am_attrvalue*, *am_length* and *am_flags* fields are used to store the value of the named attribute, and some control flags for that sub-operation, respectively. Their use varies depending on the value of the *am_opcode* field.

ATTR_OP_GET

The *am_attrvalue* field is a pointer to a empty buffer that will be overwritten with the value of the named attribute. The *am_length* field is initially the total size of the memory buffer that the *am_attrvalue* field points to. After the operation, the *am_length* field contains the actual size of the attribute´s value. The *am_flags* field may be set to the ***ATTR_ROOT*** flag. If the process has appropriate priviledges, the ROOT namespace will be searched for the named attribute, otherwise the USER namespace will be searched.

ATTR_OP_SET

The *am_attrvalue* and *am_length* fields contain the new value for the given attribute name and its length. The ***ATTR_ROOT*** flag may be set in the *am_flags* field. If the process has appropriate priviledges, the ROOT namespace will be searched for the named attribute, otherwise the USER namespace will be searched. The ***ATTR_CREATE*** and the ***ATTR_REPLACE*** flags may also be set in the *am_flags* field (but not simultaneously). If the ***ATTR_CREATE*** flag is set, the sub-operation will set the *am_error* field to EEXIST if the named attribute already exists. If the ***ATTR_REPLACE*** flag is set, the sub-operation will set the *am_error* field to

ENOATTR if the named attribute does not already exist. If neither of those two flags are set and the attribute does not exist, then the attribute will be created with the given value. If neither of those two flags are set and the attribute already exists, then the value will be replaced with the given value.

ATTR_OP_REMOVE
The *am_attrvalue* and *am_length* fields are not used and are ignored. The *am_flags* field may be set to the ***ATTR_ROOT*** flag. If the process has appropriate priviledges, the ROOT namespace will be searched for the named attribute, otherwise the USER namespace will be searched.

The *flags* argument to the **attr_multi** call is used to control following of symbolic links in the *path* argument. The default is to follow symbolic links, *flags* should be set to ATTR_DONTFOLLOW to not follow symbolic links.

attr_multi will fail if one or more of the following are true:

[ENOENT] The named file does not exist.

[EPERM] The effective user ID does not match the owner of the file and the effective user ID is not super-user.

[ENOTDIR] A component of the path prefix is not a directory.

[EACCES] Search permission is denied on a component of the path prefix.

[EINVAL] A bit other than ATTR_DONTFOLLOW was set in the *flag* argument.

[EFAULT] *Path*, or *oplist* points outside the allocated address space of the process.

[ELOOP] A path name lookup involved too many symbolic links.

[ENAMETOOLONG] The length of *path* exceeds {*MAXPATHLEN*}, or a pathname component is longer than {*MAXNAMELEN*}.

attr_multif will fail if:

[EINVAL] A bit was set in the *flag* argument, or *fd* refers to a socket, not a file.

[EFAULT] *Oplist* points outside the allocated address space of the process.

[EBADF] *Fd* does not refer to a valid descriptor.

DIAGNOSTICS

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately. Note that the individual operations listed in the *oplist* array each have their own error return fields. The *errno* variable only records the result of the *attr_multi* call itself, not the result of any of the sub-operations.

SEE ALSO

attr(1), **attr_get**(3), **attr_remove**(3), and **attr_set**(3).

NAME

attr_remove, attr_removef − remove a user attribute of a filesystem object

C SYNOPSIS

#include <attr/attributes.h>

int attr_remove (const char *_path_, const char *_attrname_, int _flags_);

int attr_removef (int _fd_, const char *_attrname_, int _flags_);

DESCRIPTION

The **attr_remove** and **attr_removef** functions provide a way to remove previously created attributes from filesystem objects.

Path points to a path name for a filesystem object, and *fd* refers to the file descriptor associated with a file. If the attribute *attrname* exists, the attribute name and value will be removed from the fileystem object. The *flags* argument can contain the following symbols bitwise OR´ed together:

ATTR_ROOT
: Look for *attrname* in the **root** address space, not in the **user** address space. (limited to use by super-user only)

ATTR_DONTFOLLOW
: Do not follow symbolic links when resolving a *path* on an **attr_remove** function call. The default is to follow symbolic links.

attr_remove will fail if one or more of the following are true:

[ENOATTR] The attribute name given is not associated with the indicated filesystem object.

[ENOENT] The named file does not exist.

[EPERM] The effective user ID does not match the owner of the file and the effective user ID is not super-user.

[ENOTDIR] A component of the path prefix is not a directory.

[EACCES] Search permission is denied on a component of the path prefix.

[EINVAL] A bit was set in the *flag* argument that is not defined for this system call.

[EFAULT] *Path* points outside the allocated address space of the process.

[ELOOP] A path name lookup involved too many symbolic links.

[ENAMETOOLONG] The length of *path* exceeds {*MAXPATHLEN*}, or a pathname component is longer than {*MAXNAMELEN*}.

attr_removef will fail if:

[ENOATTR] The attribute name given is not associated with the indicated filesystem object.

[EINVAL] A bit was set in the *flag* argument that is not defined for this system call, or *fd* refers to a socket, not a file.

[EFAULT] *Attrname* points outside the allocated address space of the process.

[EBADF] *Fd* does not refer to a valid descriptor.

DIAGNOSTICS

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

SEE ALSO

attr(1), **attr_get**(3), **attr_multi**(3), and **attr_set**(3).

NAME

attr_set, attr_setf − set the value of a user attribute of a filesystem object

C SYNOPSIS

#include <attr/attributes.h>

int attr_set (const char ****path*, const char ****attrname***,**
const char ****attrvalue*, const int** *valuelength***,**
int *flags***);**

int attr_setf (int *fd***, const char ****attrname***,**
const char ****attrvalue*, const int** *valuelength***,**
int *flags***);**

DESCRIPTION

The **attr_set** and **attr_setf** functions provide a way to create attributes and set/change their values.

Path points to a path name for a filesystem object, and *fd* refers to the file descriptor associated with a file. If the attribute *attrname* does not exist, an attribute with the given name and value will be created and associated with that indicated filesystem object. If an attribute with that name already exists on that filesystem object, the existing value is replaced with the new value given in this call. The new attribute value is copied from the *attrvalue* buffer for a total of *valuelength* bytes. The *flags* argument can contain the following symbols bitwise OR´ed together:

ATTR_ROOT
: Look for *attrname* in the **root** address space, not in the **user** address space. (limited to use by super-user only)

ATTR_DONTFOLLOW
: Do not follow symbolic links when resolving a *path* on an **attr_set** function call. The default is to follow symbolic links.

ATTR_CREATE
: Return an error (EEXIST) if an attribute of the given name already exists on the indicated filesystem object, otherwise create an attribute with the given name and value. This flag is used to implement a pure create operation, without this flag **attr_set** will create the attribute if it does not already exist. An error (EINVAL) will be returned if both ATTR_CREATE and ATTR_REPLACE are set in the same call.

ATTR_REPLACE
: Return an error (ENOATTR) if an attribute of the given name does not already exist on the indicated filesystem object, otherwise replace the existing attribute´s value with the given value. This flag is used to implement a pure replacement operation, without this flag **attr_set** will create the attribute if it does not already exist. An error (EINVAL) will be returned if both ATTR_CREATE and ATTR_REPLACE are set in the same call.

attr_set will fail if one or more of the following are true:

[ENOATTR]	The attribute name given is not associated with the indicated filesystem object and the ATTR_REPLACE flag bit was set.
[E2BIG]	The value of the given attribute is too large, it exceeds the maximum allowable size of an attribute value.
[EEXIST]	The attribute name given is already associated with the indicated filesystem object and the ATTR_CREATE flag bit was set.
[ENOENT]	The named file does not exist.
[EPERM]	The effective user ID does not match the owner of the file and the effective user ID is not super-user.
[ENOTDIR]	A component of the path prefix is not a directory.
[EACCES]	Search permission is denied on a component of the path prefix.

[EINVAL] A bit was set in the *flag* argument that is not defined for this system call, or both the ATTR_CREATE and ATTR_REPLACE flags bits were set.

[EFAULT] *Path, attrname,* or *attrvalue* points outside the allocated address space of the process.

[ELOOP] A path name lookup involved too many symbolic links.

[ENAMETOOLONG] The length of *path* exceeds {*MAXPATHLEN*}, or a pathname component is longer than {*MAXNAMELEN*}.

attr_setf will fail if:

[ENOATTR] The attribute name given is not associated with the indicated filesystem object and the ATTR_REPLACE flag bit was set.

[E2BIG] The value of the given attribute is too large, it exceeds the maximum allowable size of an attribute value.

[EEXIST] The attribute name given is already associated with the indicated filesystem object and the ATTR_CREATE flag bit was set.

[EINVAL] A bit was set in the *flag* argument that is not defined for this system call, or both the ATTR_CREATE and ATTR_REPLACE flags bits were set, or *fd* refers to a socket, not a file.

[EFAULT] *Attrname,* or *attrvalue* points outside the allocated address space of the process.

[EBADF] *Fd* does not refer to a valid descriptor.

DIAGNOSTICS

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

SEE ALSO

attr(1), **attr_get**(3), **attr_multi**(3), and **attr_remove**(3).

NAME

attr - Extended attributes

DESCRIPTION

Extended attributes are name:value pairs associated permanently with files and directories, similar to the environment strings associated with a process. An attribute may be defined or undefined. If it is defined, its value may be empty or non-empty.

Extended attributes are extensions to the normal attributes which are associated with all inodes in the system (i.e. the **stat**(2) data). They are often used to provide additional functionality to a filesystem − for example, additional security features such as Access Control Lists (ACLs) may be implemented using extended attributes.

Users with search access to a file or directory may retrieve a list of attribute names defined for that file or directory.

Extended attributes are accessed as atomic objects. Reading retrieves the whole value of an attribute and stores it in a buffer. Writing replaces any previous value with the new value.

Space consumed for extended attributes is counted towards the disk quotas of the file owner and file group.

Currently, support for extended attributes is implemented on Linux by the ext2, ext3 and XFS filesystem patches, which can be downloaded from **http://acl.bestbits.at/** and **http://oss.sgi.com/projects/xfs/** respectively.

EXTENDED ATTRIBUTE NAMESPACES

Attribute names are zero-terminated strings. The attribute name is always specified in the fully qualified *namespace.attribute* form, eg. *user.mime_type*, *trusted.md5sum*, or *system.posix_acl_access*.

The namespace mechanism is used to define different classes of extended attributes. These different classes exist for several reasons, e.g. the permissions and capabilities required for manipulating extended attributes of one namespace may differ to another.

Currently the *user*, *trusted*, and *system* extended attribute classes are defined as described below. Additional classes may be added in the future.

Extended user attributes

Extended user attributes may be assigned to files and directories for storing arbitrary additional information such as the mime type, character set or encoding of a file. The access permissions for user attributes are defined by the file permission bits.

The file permission bits of regular files and directories are interpreted differently from the file permission bits of special files and symbolic links. For regular files and directories the file permission bits define access to the file's contents, while for device special files they define access to the device described by the special file. The file permissions of symbolic links are not used in access checks. These differences would allow users to consume filesystem resources in a way not controllable by disk quotas for group or world writable special files and directories.

For this reason, extended user attributes are only allowed for regular files and directories, and access to extended user attributes is restricted to the owner and to users with appropriate capabilities for directories with the sticky bit set (see the **chmod**(1) manual page for an explanation of Sticky Directories).

Trusted extended attributes

Trusted extended attributes are visible and accessible only to processes that have the CAP_SYS_ADMIN capability (the super user usually has this capability). Attributes in this class are used to implement mechanisms in user space (i.e., outside the kernel) which keep information in extended attributes to which ordinary processes should not have access.

Extended system attributes

Extended system attributes are used by the kernel to store system objects such as Access Control Lists and Capabilities. Read and write access permissions to system attributes depend on the policy implemented for each system attribute implemented in the kernel.

FILESYSTEM DIFFERENCES

The kernel and the filesystem may place limits on the maximum number and size of extended attributes that can be associated with a file.

In the current ext2 and ext3 filesystem implementations, all extended attributes must fit on a single filesystem block (1024, 2048 or 4096 bytes, depending on the block size specified when the filesystem was created). This limit may be removed in a future version.

In the XFS filesystem implementation, there is no practical limit on the number of extended attributes associated with a file, and the algorithms used to store extended attribute information on disk are scalable (stored either inline in the inode, as an extent, or in a B+ tree).

ADDITIONAL NOTES

Since the filesystems on which extended attributes are stored might also be used on architectures with a different byte order and machine word size, care should be taken to store attribute values in an architecture independent format.

AUTHORS

Andreas Gruenbacher, <*a.gruenbacher@computer.org*> and the SGI XFS development team, <*linux-xfs@oss.sgi.com*>.

SEE ALSO

getfattr(1), setfattr(1).