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 ho w 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_fla gs* 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 b uffer 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_fla gs* field may be set to the ***ATTR_ROOT*** flag. If the process has appropriate privileges, 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***A TTR_ROOT*** flag may be set in the *am_flags* field. If the process has appropriate privileges, 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 privileges, 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_list**(3), **attr_remove**(3), **attr_set**(3)