NAME

attr_list, attr_listf − list the names of the user attributes of a filesystem object

C SYNOPSIS

#include <attr/attributes.h>

int attr_list (const char *path, char *buffer,
const int buffersize, int flags,
attrlist_cursor_t *cursor);

int attr_listf (int fd, char *buffer,
const int buffersize, int flags,
attrlist_cursor_t *cursor);

DESCRIPTION

The **attr_list** and **attr_listf** functions provide a way to list the existing attributes of a filesystem object.

Path points to a path name for a filesystem object, and *fd* refers to the file descriptor associated with a file. The *buffer* will be filled with a structure describing at least a portion of the attributes associated with the given filesystem object. *Buffer* will be overwritten with an ***attrlist_t*** structure containing a list of the attributes associated with that filesystem object, up to a maximum of *buffersize* bytes. The *buffer* must be sufficiently large to hold the appropriate data structures plus at least one maximally sized attribute name, but cannot be more than ATTR_MAX_VALUELEN (currently 64KB) bytes in length.

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

int32_t al_count; /* number of entries in attrlist */
int32_t al_more; /* T/F: more attrs (do syscall again) */
int32_t al_offset[1]; /* byte offsets of attrs [var-sized] */

The *al_count* field shows the number of attributes represented in this buffer, which is also the number of elements in the *al_offset* array. The*al_mor e* field will be non-zero if another **attr_list** call would result in more attributes. The*al_of fset* array contains the byte offset within the *buffer* of the structure describing each of the attributes, an ***attrlist_ent_t*** structure. The ***ATTR_ENTRY(buffer, index)*** macro will help with decoding the list. It takes a pointer to the *buffer* and an *index* into the *al_offset* array and returns a pointer to the corresponding ***attrlist_ent_t*** structure.

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

uint32_t a_valuelen; /* number bytes in value of attr */
char a_name[]; /* attr name (NULL terminated) */

The *a_valuelen* field shows the size in bytes of the value associated with the attribute whose name is stored in the *a_name* field. The name is a NULL terminated string.

Note that the value of the attribute cannot be obtained through this interface, the **attr_get** call should be used to get the value. The**attr_list** interf ace tells the calling process how large of a buffer it must have in order to get the attribute's value.

The *flags* argument can contain the following symbols bitwise OR'ed together:

ATTR_ROOT
: List the attributes that are 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_list** function call. The def ault is to follow symbolic links.

The *cursor* argument is a pointer to an opaque data structure that the kernel uses to track the calling process's position in the attribute list. The only valid operations on a *cursor* are to pass it into an **attr_list** function call or to zero it out. It should be zero'ed out before the first **attr_list** call. Note that multi-threaded applications may keep more than one *cursor* in order to serve multiple contexts, ie: the **attr_list**

call is "thread-safe".

attr_list 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 was set in the *flag* argument that is not defined for this system call, or the buffer was too small or too large.

[EFAULT] Either *Path* or *buffer* points outside the allocated address space of the process, or *buffer* or *bufsize* are not 32bit aligned.

[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*}.

[ENOATTR] *attribute* does not exist for this file.

attr_listf will fail if:

[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, or the buffer was too small or too large.

[EFAULT] Either *Path* or *buffer* points outside the allocated address space of the process, or *buffer* or *bufsize* are not 32bit aligned.

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

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

SEE ALSO

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