NAME

listxattr, llistxattr, flistxattr − list extended attribute names

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/xattr.h>

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

DESCRIPTION

Extended attributes are *name*:*value* pairs associated with inodes (files, directories, symbolic links, 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 **xattr**(7).

listxattr() retrieves the list of extended attribute names associated with the given *path* in the filesystem. The retrieved list is placed in *list*, a caller-allocated buffer whose size (in bytes) is specified in the argument *size*. 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 referred to by *fd* (as returned by **open**(2)) is interrogated in place of *path*.

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

If *size* is specified as zero, these calls return the current size of the list of extended attribute names (and leave *list* unchanged). This can be used to determine the size of the buffer that should be supplied in a subsequent call. (But, bear in mind that there is a possibility that the set of extended attributes may change between the two calls, so that it is still necessary to check the return status from the second call.)

Example

The *list* of names is returned as an unordered array of null-terminated character strings (attribute names are separated by null bytes ('\0')), like this:

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

Filesystems that 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 nonnegative number is returned indicating the size of the extended attribute name list. On failure, −1 is returned and *errno* is set to indicate the error.

ERRORS

E2BIG The size of the list of extended attribute names is larger than the maximum size allowed; the list cannot be retrieved. This can happen on filesystems that support an unlimited number of extended attributes per file such as XFS, for example. See BUGS.

ENOTSUP
Extended attributes are not supported by the filesystem, or are disabled.

ERANGE
The *size* of the *list* buffer is too small to hold the result.

In addition, the errors documented in **stat**(2) can also occur.

VERSIONS

These system calls have been available since Linux 2.4; glibc support is provided since glibc 2.3.

STANDARDS

These system calls are Linux-specific.

BUGS

As noted in **xattr**(7), the VFS imposes a limit of 64 kB on the size of the extended attribute name list returned by **listxattr**(). If the total size of attribute names attached to a file exceeds this limit, it is no longer possible to retrieve the list of attribute names.

EXAMPLES

The following program demonstrates the usage of **listxattr**() and **getxattr**(2). For the file whose pathname is provided as a command-line argument, it lists all extended file attributes and their values.

To keep the code simple, the program assumes that attribute keys and values are constant during the execution of the program. A production program should expect and handle changes during execution of the program. For example, the number of bytes required for attribute keys might increase between the two calls to **listxattr**(). An application could handle this possibility using a loop that retries the call (perhaps up to a predetermined maximum number of attempts) with a larger buffer each time it fails with the error **ERANGE**. Calls to**getxattr**(2) could be handled similarly .

The following output was recorded by first creating a file, setting some extended file attributes, and then listing the attributes with the example program.

Example output

```
$ touch /tmp/foo
$ setfattr −n user.fred −v chocolate /tmp/foo
$ setfattr −n user.frieda −v bar /tmp/foo
$ setfattr −n user.empty /tmp/foo
$ ./listxattr /tmp/foo
user.fred: chocolate
user.frieda: bar
user.empty: <no value>
```

Program source (listxattr.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/xattr.h>

int
main(int argc, char *argv[])
{
    char     *buf, *key, *val;
    ssize_t  buflen, keylen, vallen;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s path\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    /*
     * Determine the length of the buffer needed.
     */
    buflen = listxattr(argv[1], NULL, 0);
    if (buflen == −1) {
        perror("listxattr");
        exit(EXIT_FAILURE);
```

```
          }
          if (buflen == 0) {
              printf("%s has no attributes.\n", argv[1]);
              exit(EXIT_SUCCESS);
          }

          /*
           * Allocate the buffer.
           */
          buf = malloc(buflen);
          if (buf == NULL) {
              perror("malloc");
              exit(EXIT_FAILURE);
          }

          /*
           * Copy the list of attribute keys to the buffer.
           */
          buflen = listxattr(argv[1], buf, buflen);
          if (buflen == −1) {
              perror("listxattr");
              exit(EXIT_FAILURE);
          }

          /*
           * Loop over the list of zero terminated strings with the
           * attribute keys. Use the remaining buffer length to determine
           * the end of the list.
           */
          key = buf;
          while (buflen > 0) {

              /*
               * Output attribute key.
               */
              printf("%s: ", key);

              /*
               * Determine length of the value.
               */
              vallen = getxattr(argv[1], key, NULL, 0);
              if (vallen == −1)
                  perror("getxattr");

              if (vallen > 0) {

                  /*
                   * Allocate value buffer.
                   * One extra byte is needed to append 0x00.
                   */
                  val = malloc(vallen + 1);
                  if (val == NULL) {
                      perror("malloc");
                      exit(EXIT_FAILURE);
```

```
                }

                /*
                 * Copy value to buffer.
                 */
                vallen = getxattr(argv[1], key, val, vallen);
                if (vallen == −1) {
                    perror("getxattr");
                } else {
                    /*
                     * Output attribute value.
                     */
                    val[vallen] = 0;
                    printf("%s", val);
                }

                free(val);
            } else if (vallen == 0) {
                printf("<no value>");
            }

            printf("\n");

            /*
             * Forward to next attribute key.
             */
            keylen = strlen(key) + 1;
            buflen −= keylen;
            key += keylen;
        }

        free(buf);
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

getfattr(1), **setfattr**(1), **getxattr**(2), **open**(2), **removexattr**(2), **setxattr**(2), **stat**(2), **symlink**(7), **xattr**(7)