NAME

readlink − read value of a symbolic link

SYNOPSIS

#include <unistd.h>

ssize_t readlink(const char * *path*, **char *** *buf*, **size_t** *bufsiz*);

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

readlink():
_BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
_XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED || _POSIX_C_SOURCE >= 200112L

DESCRIPTION

readlink() places the contents of the symbolic link *path* in the buffer *buf*, which has size *bufsiz*. **readlink**() does not append a null byte to *buf*. It will truncate the contents (to a length of *bufsiz* characters), in case the buffer is too small to hold all of the contents.

RETURN VALUE

On success, **readlink**() returns the number of bytes placed in *buf*. On error, −1 is returned and *errno* is set to indicate the error.

ERRORS

EACCES
Search permission is denied for a component of the path prefix. (See also **path_resolution**(7).)

EFAULT
buf extends outside the process's allocated address space.

EINVAL
bufsiz is not positive.

EINVAL
The named file is not a symbolic link.

EIO An I/O error occurred while reading from the file system.

ELOOP
Too many symbolic links were encountered in translating the pathname.

ENAMETOOLONG
A pathname, or a component of a pathname, was too long.

ENOENT
The named file does not exist.

ENOMEM
Insufficient kernel memory was available.

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

CONFORMING TO

4.4BSD (**readlink**() first appeared in 4.2BSD), POSIX.1-2001.

NOTES

In versions of glibc up to and including glibc 2.4, the return type of **readlink**() was declared as *int*. Nowadays, the return type is declared as *ssize_t*, as (newly) required in POSIX.1-2001.

Using a statically sized buffer might not provide enough room for the symbolic link contents. The required size for the buffer can be obtained from the *stat.st_size* value returned by a call to **lstat**(2) on the link. However, the number of bytes written by **readlink**() should be checked to make sure that the size of the

symbolic link did not increase between the calls. Dynamically allocating the buffer for **readlink**() also addresses a common portability problem when using *PATH_MAX* for the buffer size, as this constant is not guaranteed to be defined per POSIX if the system does not have such limit.

EXAMPLE

The following program allocates the buffer needed by **readlink**() dynamically from the information provided by **lstat**(), making sure there's no race condition between the calls.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct stat sb;
    char *linkname;
    ssize_t r;

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

    if (lstat(argv[1], &sb) == -1) {
        perror("lstat");
        exit(EXIT_FAILURE);
    }

    linkname = malloc(sb.st_size + 1);
    if (linkname == NULL) {
        fprintf(stderr, "insufficient memory\n");
        exit(EXIT_FAILURE);
    }

    r = readlink(argv[1], linkname, sb.st_size + 1);

    if (r == -1) {
        perror("lstat");
        exit(EXIT_FAILURE);
    }

    if (r > sb.st_size) {
        fprintf(stderr, "symlink increased in size "
                        "between lstat() and readlink()\n");
        exit(EXIT_FAILURE);
    }

    linkname[r] = '\0';

    printf("'%s' points to '%s'\n", argv[1], linkname);

    exit(EXIT_SUCCESS);
```

```
}
```

SEE ALSO

readlink(1), **lstat**(2), **readlinkat**(2), **stat**(2), **symlink**(2), **path_resolution**(7), **symlink**(7)

COLOPHON

This page is part of release 3.53 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man−pages/.