link - Man Page

link one file to another file

Prolog

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

Synopsis

```
#include <unistd.h>

int link(const char *path1, const char *path2);

#include <fcntl.h>

int linkat(int fd1, const char *path1, int fd2,
    const char *path2, int flag);
```

Description

The *link*() function shall create a new link (directory entry) for the existing file, *path1*.

The *path1* argument points to a pathname naming an existing file. The *path2* argument points to a pathname naming the new directory entry to be created. The *link*() function shall atomically create a new link for the existing file and the link count of the file shall be incremented by one.

link - Man Page

directories.

If *path1* names a symbolic link, it is implementation-defined whether *link*() follows the symbolic link, or creates a new link to the symbolic link itself.

Upon successful completion, *link*() shall mark for update the last file status change timestamp of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

If *link*() fails, no link shall be created and the link count of the file shall remain unchanged.

The implementation may require that the calling process has permission to access the existing file.

The *linkat*() function shall be equivalent to the *link*() function except that symbolic links shall be handled as specified by the value of *flag* (see below) and except in the case where either *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *path2* and the file descriptor *fd2*. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_SYMLINK_FOLLOW

If *path1* names a symbolic link, a new link for the target of the symbolic link is created.

link - Man Page

parameter, the current working directory shall be used for the respective *path* argument. If both *fd1* and *fd2* have value AT_FDCWD, the behavior shall be identical to a call to *link*(), except that symbolic links shall be handled as specified by the value of *flag*.

If the AT_SYMLINK_FOLLOW flag is clear in the *flag* argument and the *path1* argument names a symbolic link, a new link is created for the symbolic link *path1* and not its target.

Return Value

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set *errno* to indicate the error.

Errors

These functions shall fail if:

EACCES

A component of either path prefix denies search permission, or the requested link requires writing in a directory that denies write permission, or the calling process does not have permission to access the existing file and this is required by the implementation.

EEXIST

The *path2* argument resolves to an existing directory entry or refers to a symbolic link.

ELOOP

A loop exists in symbolic links encountered during resolution of the *path1* or *path2* argument.

link - Man Page

{LINK_MAX}.

ENAMETOOLONG

The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT

A component of either path prefix does not exist; the file named by *path1* does not exist; or *path1* or *path2* points to an empty string.

ENOENT or **ENOTDIR**

The *path1* argument names an existing non-directory file, and the *path2* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path2* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC

The directory to contain the link cannot be extended.

ENOTDIR

A component of either path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path1* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory, or the *path1* argument names an existing non-directory file and the *path2* argument names a nonexistent file, contains at least one non-<slash> character, and ends with one or more trailing <slash> characters.

EPERM

implementation prohibits using *link*() on directories.

EROFS

The requested link requires writing in a directory on a read-only file system.

EXDEV

The link named by *path2* and the file named by *path1* are on different file systems and the implementation does not support links between file systems.

EXDEV

path1 refers to a named STREAM.

The *linkat*() function shall fail if:

EACCES

The access mode of the open file description associated with *fd1* or *fd2* is not O_SEARCH and the permissions of the directory underlying *fd1* or *fd2*, respectively, do not permit directory searches.

EBADF

The *path1* or *path2* argument does not specify an absolute path and the *fd1* or *fd2* argument, respectively, is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR

The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*, respectively, is a file descriptor associated with a non-directory file.

link - Man Page

More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path1* or *path2* argument.

ENAMETOOLONG

The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *linkat*() function may fail if:

EINVAL

The value of the *flag* argument is not valid.

The following sections are informative.

Examples

Creating a Link to a File

The following example shows how to create a link to a file named **/home/cnd/mod1** by creating a new directory entry named **/modules/pass1**.

```
#include <unistd.h>

char *path1 = "/home/cnd/mod1";
char *path2 = "/modules/pass1";
int   status;
...
status = link (path1, path2);
```

link - Man Page

In the following program example, the *link*() function links the **/etc/passwd** file (defined as **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the current password file. Then, after removing the current password file (defined as **PASSWDFILE**), the new password file is saved as the current password file using the *link*() function again.

```
#include <unistd.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
#define SAVEFILE "/etc/opasswd"
...
/* Save current password file */
link (PASSWDFILE, SAVEFILE);

/* Remove current password file. */
unlink (PASSWDFILE);

/* Save new password file as current password file. */
link (LOCKFILE,PASSWDFILE);
```

Application Usage

Some implementations do allow links between file systems.

If *path1* refers to a symbolic link, application developers should use *linkat*() with appropriate flags to select whether or not the symbolic link should be resolved.

Rationale

Linking to a directory is restricted to the superuser in most historical implementations because this capability may produce loops in *unlink*() from doing this. Other functions could do it if the implementor designed such an extension.

Some historical implementations allow linking of files on different file systems. Wording was added to explicitly allow this optional behavior.

The exception for cross-file system links is intended to apply only to links that are programmatically indistinguishable from “hard” links.

The purpose of the *linkat*() function is to link files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *link*(), resulting in unspecified behavior. By opening a file descriptor for the directory of both the existing file and the target location and using the *linkat*() function it can be guaranteed that the both filenames are in the desired directories.

The AT_SYMLINK_FOLLOW flag allows for implementing both common behaviors of the *link*() function. The POSIX specification requires that if *path1* is a symbolic link, a new link for the target of the symbolic link is created. Many systems by default or as an alternative provide a mechanism to avoid the implicit symbolic link lookup and create a new link for the symbolic link itself.

Earlier versions of this standard specified only the *link*() function, and required it to behave like *linkat*() with the AT_SYMLINK_FOLLOW flag. However, historical practice from SVR4 and Linux kernels had *link*() behaving like *linkat*() with no flags, and many systems that attempted to provide a conforming *link*() function did so in a way that was rarely used, and when it was used did not conform to the standard (e.g., by not being atomic, or by dereferencing the symbolic link incorrectly). Since applications could not rely on *link*() following

link - Man Page

Future Directions

None.

See Also

rename(), symlink(), unlink()

The Base Definitions volume of POSIX.1-2017, <fcntl.h>, <unistd.h>

Copyright

Referenced By

link - Man Page

2017 IEEE/The Open Group POSIX Programmer's Manual