NAME

rename - change the name of a file

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <stdio.h>

int
rename(*const char *from*, *const char *to*);

int
renameat(*int fromfd*, *const char *from*, *int tofd*, *const char *to*);

DESCRIPTION

The **rename**() system call causes the link named *from* to be renamed as *to*. If *to* exists, it is first removed. Both *from* and *to* must be of the same type (that is, both directories or both non-directories), and must reside on the same file system.

The **rename**() system call guarantees that if *to* already exists, an instance of *to* will always exist, even if the system should crash in the middle of the operation.

If the final component of *from* is a symbolic link, the symbolic link is renamed, not the file or directory to which it points.

If *from* and *to* resolve to the same directory entry, or to different directory entries for the same existing file, **rename**() returns success without taking any further action.

The **renameat**() system call is equivalent to **rename**() except in the case where either *from* or *to* specifies a relative path. If *from* is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor *fromfd* instead of the current working directory. If the *to* is a relative path, the same happens only relative to the directory associated with *tofd*. If the **renameat**() is passed the special value AT_FDCWD in the *fromfd* or *tofd* parameter, the current working directory is used in the determination of the file for the respective path parameter.

RETURN VALUES

The **rename**() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **rename**() system call will fail and neither of the argument files will be affected if:

[ENAMETOOLONG] A component of either pathname exceeded 255 characters, or the entire length of either path name exceeded 1023 characters.

[ENOENT] A component of the *from* path does not exist, or a path prefix of *to* does not exist.

[EACCES] A component of either path prefix denies search permission.

[EACCES] The requested link requires writing in a directory with a mode that denies write permission.

[EACCES] The directory pointed at by the *from* argument denies write permission, and the operation would move it to another parent directory.

[EPERM] The file pointed at by the *from* argument has its immutable, undeletable or append-only flag set, see the chflags(2) manual page for more information.

[EPERM] The parent directory of the file pointed at by the *from* argument has its immutable or append-only flag set.

[EPERM] The parent directory of the file pointed at by the *to* argument has its immutable flag set.

[EPERM] The directory containing *from* is marked sticky, and neither the containing directory nor *from* are owned by the effective user ID.

[EPERM] The file pointed at by the *to* argument exists, the directory containing *to* is marked sticky, and neither the containing directory nor *to* are owned by the effective user ID.

[ELOOP] Too many symbolic links were encountered in translating either pathname.

[ENOTDIR] A component of either path prefix is not a directory.

[ENOTDIR] The *from* argument is a directory, but *to* is not a directory.

[EISDIR] The *to* argument is a directory, but *from* is not a directory.

[EXDEV] The link named by *to* and the file named by *from* are on different logical devices (file systems). Note that this error code will not be returned if the implementation permits cross-device links.

[ENOSPC] The directory in which the entry for the new name is being placed cannot be extended because there is no space left on the file system containing the directory.

[EDQUOT] The directory in which the entry for the new name is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted.

[EIO] An I/O error occurred while making or updating a directory entry.

[EINTEGRITY] Corrupted data was detected while reading from the file system.

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

[EFAULT] Path points outside the process's allocated address space.

[EINVAL] The *from* argument is a parent directory of *to*, or an attempt is made to rename '.' or '..'.

[ENOTEMPTY] The *to* argument is a directory and is not empty.

[ECAPMODE] **rename**() was called and the process is in capability mode.

In addition to the errors returned by the **rename**(), the **renameat**() may fail if:

[EBADF] The *from* argument does not specify an absolute path and the *fromfd* argument is neither AT_FDCWD nor a valid file descriptor open for searching, or the *to* argument does not specify an absolute path and the *tofd* argument is neither AT_FDCWD nor a valid file descriptor open for searching.

[ENOTDIR] The *from* argument is not an absolute path and *fromfd* is neither AT_FDCWD nor a file descriptor associated with a directory, or the *to* argument is not an absolute path and *tofd* is neither AT_FDCWD nor a file descriptor associated with a directory.

[ECAPMODE] AT_FDCWD is specified and the process is in capability mode.

[ENOTCAPABLE] *path* is an absolute path or contained a ".." component leading to a directory outside of the directory hierarchy specified by *fromfd* or *tofd*.

[ENOTCAPABLE] The *fromfd* file descriptor lacks the CAP_RENAMEAT_SOURCE right, or the *tofd* file descriptor lacks the CAP_RENAMEAT_TARGET right.

SEE ALSO

chflags(2), open(2), symlink(7)

STANDARDS

The **rename**() system call is expected to conform to ISO/IEC 9945-1:1996 ("POSIX.1"). The **renameat**() system call follows The Open Group Extended API Set 2 specification.

HISTORY

The **renameat**() system call appeared in FreeBSD 8.0.