Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'unlink.3p' command

$ man unlink.3p

UNLINK(3P) POSIX Programmer's Manual UNLINK(3P)

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.

NAME

unlink, unlinkat ? remove a directory entry

SYNOPSIS

```
#include <unistd.h>
int unlink(const char *path);
#include <fcntl.h>
int unlinkat(int fd, const char *path, int flag);
```

DESCRIPTION

The unlink() function shall remove a link to a file. If path names a symbolic link, unlink() shall remove the symbolic link named by path and shall not affect any file or directory named by the contents of the symbolic link. Otherwise, unlink() shall remove the link named by the pathname pointed to by path and shall decrement the link count of the file referenced by the link.

When the file's link count becomes 0 and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when

the last link is removed, the link shall be removed before unlink() re? turns, but the removal of the file contents shall be postponed until all references to the file are closed.

The path argument shall not name a directory unless the process has ap? propriate privileges and the implementation supports using unlink() on directories.

Upon successful completion, unlink() shall mark for update the last data modification and last file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last file sta? tus change timestamp of the file shall be marked for update.

The unlinkat() function shall be equivalent to the unlink() or rmdir() function except in the case where path specifies a relative path. In this case the directory entry to be removed is determined relative to the directory associated with the file descriptor fd instead of the current working directory. If the access mode of the open file descrip? tion 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 ac? cess 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_REMOVEDIR

Remove the directory entry specified by fd and path as a direc? tory, not a normal file.

If unlinkat() is passed the special value AT_FDCWD in the fd parameter, the current working directory shall be used and the behavior shall be identical to a call to unlink() or rmdir() respectively, depending on whether or not the AT_REMOVEDIR bit is set in flag.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set errno to indicate the error. If -1 is returned, the named file shall not be changed.

ERRORS

These functions shall fail and shall not unlink the file if:

EACCES Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the directory entry to be removed.

EBUSY The file named by the path argument cannot be unlinked because it is being used by the system or another process and the imple? mentation considers this an error.

ELOOP A loop exists in symbolic links encountered during resolution of the path argument.

ENAMETOOLONG

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

ENOENT A component of path does not name an existing file or path is an empty string.

ENOTDIR

A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the path 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 di? rectory nor a symbolic link to a directory.

EPERM The file named by path is a directory, and either the calling process does not have appropriate privileges, or the implementa? tion prohibits using unlink() on directories.

EPERM or EACCES

The S_ISVTX flag is set on the directory containing the file re? ferred to by the path argument and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1?2017, Section 4.3, Directory Protection.

EROFS The directory entry to be unlinked is part of a read-only file system.

The unlinkat() function shall fail if:

EACCES The access mode of the open file description associated with fd

is not O_SEARCH and the permissions of the directory underlying fd do not permit directory searches.

EBADF The path argument does not specify an absolute path and the fd argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR

The path argument is not an absolute path and fd is a file de? scriptor associated with a non-directory file.

EEXIST or ENOTEMPTY

The flag parameter has the AT_REMOVEDIR bit set and the path ar? gument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.

ENOTDIR

The flag parameter has the AT_REMOVEDIR bit set and path does not name a directory.

These functions may fail and not unlink the file if:

EBUSY The file named by path is a named STREAM.

ELOOP More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path argument.

ENAMETOOLONG

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

ETXTBSY

The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.

The unlinkat() function may fail if:

EINVAL The value of the flag argument is not valid.

The following sections are informative.

EXAMPLES

Removing a Link to a File

The following example shows how to remove a link to a file named

/home/cnd/mod1 by removing the entry named /modules/pass1.

```
#include <unistd.h>
char *path = "/modules/pass1";
int   status;
...
status = unlink(path);
```

Checking for an Error

The following example fragment creates a temporary password lock file named LOCKFILE, which is defined as /etc/ptmp, and gets a file descrip? tor for it. If the file cannot be opened for writing, unlink() is used to remove the link between the file descriptor and LOCKFILE.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>
#define LOCKFILE "/etc/ptmp"
int pfd;  /* Integer for file descriptor returned by open call. */
FILE *fpfd;  /* File pointer for use in putpwent(). */
...
/* Open password Lock file. If it exists, this is an error. */
if ((pfd = open(LOCKFILE, O_WRONLY| O_CREAT | O_EXCL, S_IRUSR
    | S_IWUSR | S_IRGRP | S_IROTH)) == -1)  {
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
/* Lock file created; proceed with fdopen of lock file so that
   putpwent() can be used.
 */
if ((fpfd = fdopen(pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
```

```
        exit(1);
    }
```

Replacing Files

The following example fragment uses unlink() to discard links to files, so that they can be replaced with new versions of the files. The first call removes the link to LOCKFILE if an error occurs. Successive calls remove the links to SAVEFILE and PASSWDFILE so that new links can be created, then removes the link to LOCKFILE when it is no longer needed.

```
    #include <sys/types.h>
    #include <stdio.h>
    #include <fcntl.h>
    #include <errno.h>
    #include <unistd.h>
    #include <sys/stat.h>
    #define LOCKFILE "/etc/ptmp"
    #define PASSWDFILE "/etc/passwd"
    #define SAVEFILE "/etc/opasswd"
    ...
    /* If no change was made, assume error and leave passwd unchanged. */
    if (!valid_change) {
        fprintf(stderr, "Could not change password for user %s\n", user);
        unlink(LOCKFILE);
        exit(1);
    }
    /* Change permissions on new password file. */
    chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);
    /* Remove saved password file. */
    unlink(SAVEFILE);
    /* Save current password file. */
    link(PASSWDFILE, SAVEFILE);
    /* Remove current password file. */
    unlink(PASSWDFILE);
    /* Save new password file as current password file. */
```

```
link(LOCKFILE,PASSWDFILE);
/* Remove lock file. */
unlink(LOCKFILE);
exit(0);
```

APPLICATION USAGE

Applications should use rmdir() to remove a directory.

RATIONALE

Unlinking a directory is restricted to the superuser in many historical implementations for reasons given in link() (see also rename()).

The meaning of [EBUSY] in historical implementations is ``mount point busy''. Since this volume of POSIX.1?2017 does not cover the system ad? ministration concepts of mounting and unmounting, the description of the error was changed to ``resource busy''. (This meaning is used by some device drivers when a second process tries to open an exclusive use device.) The wording is also intended to allow implementations to refuse to remove a directory if it is the root or current working di? rectory of any process.

The standard developers reviewed TR 24715?2006 and noted that LSB-con? forming implementations may return [EISDIR] instead of [EPERM] when un? linking a directory. A change to permit this behavior by changing the requirement for [EPERM] to [EPERM] or [EISDIR] was considered, but de? cided against since it would break existing strictly conforming and conforming applications. Applications written for portability to both POSIX.1?2008 and the LSB should be prepared to handle either error code.

The purpose of the unlinkat() function is to remove directory entries in directories other than the current working directory without expo? sure to race conditions. Any part of the path of a file could be changed in parallel to a call to unlink(), resulting in unspecified be? havior. By opening a file descriptor for the target directory and using the unlinkat() function it can be guaranteed that the removed directory entry is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

close(), link(), remove(), rename(), rmdir(), symlink()

The Base Definitions volume of POSIX.1?2017, Section 4.3, Directory Protection, <fcntl.h>, <unistd.h>