Rocky Enterprise Linux 9.2 Manual Pages on command 'nftw.3'

$ man nftw.3

FTW(3) Linux Programmer's Manual FTW(3)

NAME

ftw, nftw - file tree walk

SYNOPSIS

```
    #include <ftw.h>

    int nftw(const char *dirpath,

            int (*fn) (const char *fpath, const struct stat *sb,

                       int typeflag, struct FTW *ftwbuf),

            int nopenfd, int flags);

    #include <ftw.h>

    int ftw(const char *dirpath,

            int (*fn) (const char *fpath, const struct stat *sb,

                       int typeflag),

            int nopenfd);
```

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

nftw(): _XOPEN_SOURCE >= 500

DESCRIPTION

nftw() walks through the directory tree that is located under the directory dirpath, and calls fn() once for each entry in the tree. By default, directories are handled before the files and subdirectories they contain (preorder traversal).

To avoid using up all of the calling process's file descriptors, nopenfd specifies the maximum number of directories that nftw() will hold open simultaneously. When the search depth exceeds this, nftw() will become slower because directories have to be closed and reopened. nftw() uses at most one file descriptor for each level in the directory tree.

For each entry found in the tree, nftw() calls fn() with four arguments: fpath, sb, type?
flag, and ftwbuf. fpath is the pathname of the entry, and is expressed either as a path?
name relative to the calling process's current working directory at the time of the call
to nftw(), if dirpath was expressed as a relative pathname, or as an absolute pathname, if
dirpath was expressed as an absolute pathname. sb is a pointer to the stat structure re?
turned by a call to stat(2) for fpath.

The typeflag argument passed to fn() is an integer that has one of the following values:

FTW_F fpath is a regular file.

FTW_D fpath is a directory.

FTW_DNR

fpath is a directory which can't be read.

FTW_DP fpath is a directory, and FTW_DEPTH was specified in flags. (If FTW_DEPTH was not
specified in flags, then directories will always be visited with typeflag set to
FTW_D.) All of the files and subdirectories within fpath have been processed.

FTW_NS The stat(2) call failed on fpath, which is not a symbolic link. The probable cause
for this is that the caller had read permission on the parent directory, so that
the filename fpath could be seen, but did not have execute permission, so that the
file could not be reached for stat(2). The contents of the buffer pointed to by sb
are undefined.

FTW_SL fpath is a symbolic link, and FTW_PHYS was set in flags.

FTW_SLN

fpath is a symbolic link pointing to a nonexistent file. (This occurs only if
FTW_PHYS is not set.) In this case the sb argument passed to fn() contains infor?
mation returned by performing lstat(2) on the "dangling" symbolic link. (But see
BUGS.)

The fourth argument (ftwbuf) that nftw() supplies when calling fn() is a pointer to a
structure of type FTW:

```
struct FTW {
    int base;
    int level;
};
```

base is the offset of the filename (i.e., basename component) in the pathname given in fpath. level is the depth of fpath in the directory tree, relative to the root of the tree (dirpath, which has depth 0).

To stop the tree walk, fn() returns a nonzero value; this value will become the return value of nftw(). As long as fn() returns 0, nftw() will continue either until it has tra? versed the entire tree, in which case it will return zero, or until it encounters an error (such as a malloc(3) failure), in which case it will return -1.

Because nftw() uses dynamic data structures, the only safe way to exit out of a tree walk is to return a nonzero value from fn(). To allow a signal to terminate the walk without causing a memory leak, have the handler set a global flag that is checked by fn(). Don't use longjmp(3) unless the program is going to terminate.

The flags argument of nftw() is formed by ORing zero or more of the following flags:

FTW_ACTIONRETVAL (since glibc 2.3.3)

If this glibc-specific flag is set, then nftw() handles the return value from fn() differently. fn() should return one of the following values:

FTW_CONTINUE

Instructs nftw() to continue normally.

FTW_SKIP_SIBLINGS

If fn() returns this value, then siblings of the current entry will be skipped, and processing continues in the parent.

FTW_SKIP_SUBTREE

If fn() is called with an entry that is a directory (typeflag is FTW_D), this return value will prevent objects within that directory from being passed as arguments to fn(). nftw() continues processing with the next sib? ling of the directory.

FTW_STOP

Causes nftw() to return immediately with the return value FTW_STOP.

Other return values could be associated with new actions in the future; fn() should not return values other than those listed above.

The feature test macro _GNU_SOURCE must be defined (before including any header files) in order to obtain the definition of FTW_ACTIONRETVAL from <ftw.h>.

FTW_CHDIR

If set, do a chdir(2) to each directory before handling its contents. This is use? ful if the program needs to perform some action in the directory in which fpath re?

sides. (Specifying this flag has no effect on the pathname that is passed in the fpath argument of fn.)

FTW_DEPTH

If set, do a post-order traversal, that is, call fn() for the directory itself af? ter handling the contents of the directory and its subdirectories. (By default, each directory is handled before its contents.)

FTW_MOUNT

If set, stay within the same filesystem (i.e., do not cross mount points).

FTW_PHYS

If set, do not follow symbolic links. (This is what you want.) If not set, sym? bolic links are followed, but no file is reported twice.

If FTW_PHYS is not set, but FTW_DEPTH is set, then the function fn() is never called for a directory that would be a descendant of itself.

ftw()

ftw() is an older function that offers a subset of the functionality of nftw(). The no? table differences are as follows:

* ftw() has no flags argument. It behaves the same as when nftw() is called with flags specified as zero.

* The callback function, fn(), is not supplied with a fourth argument.

* The range of values that is passed via the typeflag argument supplied to fn() is smaller: just FTW_F, FTW_D, FTW_DNR, FTW_NS, and (possibly) FTW_SL.

RETURN VALUE

These functions return 0 on success, and -1 if an error occurs.

If fn() returns nonzero, then the tree walk is terminated and the value returned by fn() is returned as the result of ftw() or nftw().

If nftw() is called with the FTW_ACTIONRETVAL flag, then the only nonzero value that should be used by fn() to terminate the tree walk is FTW_STOP, and that value is returned as the result of nftw().

VERSIONS

nftw() is available under glibc since version 2.1.

ATTRIBUTES

For an explanation of the terms used in this section, see attributes(7).

??

```
?Interface ? Attribute     ? Value       ?
??????????????????????????????????????????
?nftw()    ? Thread safety ? MT-Safe cwd ?
??????????????????????????????????????????
?ftw()     ? Thread safety ? MT-Safe     ?
??????????????????????????????????????????
```

CONFORMING TO

POSIX.1-2001, POSIX.1-2008, SVr4, SUSv1. POSIX.1-2008 marks ftw() as obsolete.

NOTES

POSIX.1-2008 notes that the results are unspecified if fn does not preserve the current working directory.

The function nftw() and the use of FTW_SL with ftw() were introduced in SUSv1.

In some implementations (e.g., glibc), ftw() will never use FTW_SL, on other systems FTW_SL occurs only for symbolic links that do not point to an existing file, and again on other systems ftw() will use FTW_SL for each symbolic link. If fpath is a symbolic link and stat(2) failed, POSIX.1-2008 states that it is undefined whether FTW_NS or FTW_SL is passed in typeflag. For predictable results, use nftw().

BUGS

According to POSIX.1-2008, when the typeflag argument passed to fn() contains FTW_SLN, the buffer pointed to by sb should contain information about the dangling symbolic link (ob?tained by calling lstat(2) on the link). Early glibc versions correctly followed the POSIX specification on this point. However, as a result of a regression introduced in glibc 2.4, the contents of the buffer pointed to by sb were undefined when FTW_SLN is passed in typeflag. (More precisely, the contents of the buffer were left unchanged in this case.) This regression was eventually fixed in glibc 2.30, so that the glibc imple?mentation (once more) follows the POSIX specification.

EXAMPLES

The following program traverses the directory tree under the path named in its first com?mand-line argument, or under the current directory if no argument is supplied. It dis?plays various information about each file. The second command-line argument can be used to specify characters that control the value assigned to the flags argument when calling nftw().

Program source

```
#define _XOPEN_SOURCE 500
#include <ftw.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
static int
display_info(const char *fpath, const struct stat *sb,
             int tflag, struct FTW *ftwbuf)
{
    printf("%-3s %2d ",
            (tflag == FTW_D) ?   "d"   : (tflag == FTW_DNR) ? "dnr" :
            (tflag == FTW_DP) ?  "dp"  : (tflag == FTW_F) ?   "f" :
            (tflag == FTW_NS) ?  "ns"  : (tflag == FTW_SL) ?  "sl" :
            (tflag == FTW_SLN) ? "sln" : "???",
            ftwbuf->level);
    if (tflag == FTW_NS)
        printf("-------");
    else
        printf("%7jd", (intmax_t) sb->st_size);
    printf("   %-40s %d %s\n",
            fpath, ftwbuf->base, fpath + ftwbuf->base);
    return 0;           /* To tell nftw() to continue */
}
int
main(int argc, char *argv[])
{
    int flags = 0;
    if (argc > 2 && strchr(argv[2], 'd') != NULL)
        flags |= FTW_DEPTH;
    if (argc > 2 && strchr(argv[2], 'p') != NULL)
        flags |= FTW_PHYS;
    if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags)
            == -1) {
        perror("nftw");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

stat(2), fts(3), readdir(3)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.

Linux 2020-06-09 FTW(3)