NAME

ftw, nftw - file tree walk

SYNOPSIS

```
#include <ftw.h>

int ftw(const char *dirpath,
 int (*fn) (const char *fpath, const struct stat *sb,
 int typeflag),
 int nopenfd);

#define _XOPEN_SOURCE 500/* See feature_test_macros(7)
*/
#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);
```

DESCRIPTION

ftw() 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 **ftw**() will hold open simultaneously. When the search depth exceeds this, **ftw**() will become slower because directories have to be closed and reopened. **ft w**() uses at most one file descriptor for each level in the directory tree.

For each entry found in the tree, **ftw**() calls *fn*() with three arguments: *fpath*, *sb*, and *typeflag*. *fpath* is the pathname of the entry, and is expressed either as a pathname relative to the calling process's current working directory at the time of the call to **ftw**(), 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 returned by a call to stat(2) for *fpath*. *typeflag* 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_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).

  If *fpath* is a symbolic link and stat(2) failed, POSIX.1-2001 states that it is undefined whether **FTW_NS** or **FTW_SL** (see below) is passed in *typeflag*.

To stop the tree walk, *fn*() returns a nonzero value; this value will become the return value of **ftw**(). As long as*fn*() returns 0, **ft w**() will continue either until it has traversed 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 **ftw**() 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.

nftw()

The function **nftw**() is the same as **ftw**(), except that it has one additional argument, *flags*, and calls *fn*() with one more argument, *ftwbuf*.

This *flags* argument 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 sibling 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 useful if the program needs to perform some action in the directory in which *fpath* resides.

FTW_DEPTH

If set, do a post-order traversal, that is, call *fn*() for the directory itself *after* 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, symbolic 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.

For each entry in the directory tree, **nftw**() calls *fn*() with four arguments. *fpath* and *sb* are as for **ftw**(). *typeflag* may receive any of the same values as with **ftw**(), or any of the following values:

FTW_DP

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

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.)

The fourth argument that **nftw**() supplies when calling *fn*() is 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).

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.

CONFORMING TO

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

NOTES

POSIX.1-2001 note 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.

On some systems **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. For predictable control, use **nftw**().

FTW_F is returned for all objects (files, symbolic links, FIFOs, etc.) that can be stat'ed but are not a directory.

FTW_ACTIONRETVAL is glibc-specific.

EXAMPLE

The following program traverses the directory tree under the path named in its first command-line argument, or under the current directory if no argument is supplied. It displays 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 %7jd %-40s %d %sn,
(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, (intmax_t) sb->st_size,
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 3.74 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 http://www.kernel.org/doc/man-pages/.