NAME

ftw, **nftw** - traverse (walk) a file tree

SYNOPSIS

#include <ftw.h>

int
ftw(*const char *path, int (*fn)(const char *, const struct stat *, int), int maxfds*);

int
nftw(*const char *path, int (*fn)(const char *, const struct stat *, int, struct FTW *), int maxfds, int flags*);

DESCRIPTION

The **ftw**() and **nftw**() functions traverse (walk) the directory hierarchy rooted in *path*. For each object in the hierarchy, these functions call the function pointed to by *fn*. The **ftw**() function passes this function a pointer to a NUL-terminated string containing the name of the object, a pointer to a *stat* structure corresponding to the object, and an integer flag. The **nftw**() function passes the aforementioned arguments plus a pointer to a *FTW* structure as defined by <*ftw.h*> (shown below):

```
struct FTW {
   int base;          /* offset of basename into pathname */
   int level;         /* directory depth relative to starting point */
};
```

Possible values for the flag passed to *fn* are:

FTW_F A regular file.

FTW_D A directory being visited in pre-order.

FTW_DNR A directory which cannot be read. The directory will not be descended into.

FTW_DP A directory being visited in post-order (**nftw**() only).

FTW_NS A file for which no stat(2) information was available. The contents of the *stat* structure are undefined.

FTW_SL A symbolic link.

FTW_SLN A symbolic link with a non-existent target (**nftw**() only).

The **ftw**() function traverses the tree in pre-order. That is, it processes the directory before the directory's contents.

The *maxfds* argument specifies the maximum number of file descriptors to keep open while traversing the tree. It has no effect in this implementation.

The **nftw**() function has an additional *flags* argument with the following possible values:

FTW_PHYS Physical walk, do not follow symbolic links.

FTW_MOUNT The walk will not cross a mount point.

FTW_DEPTH Process directories in post-order. Contents of a directory are visited before the directory itself. By default, **nftw**() traverses the tree in pre-order.

FTW_CHDIR Change to a directory before reading it. By default, **nftw**() will change its starting directory. The current working directory will be restored to its original value before **nftw**() returns.

RETURN VALUES

If the tree was traversed successfully, the **ftw**() and **nftw**() functions return 0. If the function pointed to by *fn* returns a non-zero value, **ftw**() and **nftw**() will stop processing the tree and return the value from *fn*. Both functions return -1 if an error is detected.

EXAMPLES

Following there is an example that shows how **nftw** can be used. It traverses the file tree starting at the directory pointed by the only program argument and shows the complete path and a brief indicator about the file type.

```
#include <ftw.h>
#include <stdio.h>
#include <sysexits.h>

int
nftw_callback(const char *path, const struct stat *sb, int typeflag, struct FTW *ftw)
{
        char type;

        switch(typeflag) {
        case FTW_F:
```

```
                type = 'F';
                break;
        case FTW_D:
                type = 'D';
                break;
        case FTW_DNR:
                type = '-';
                break;
        case FTW_DP:
                type = 'd';
                break;
        case FTW_NS:
                type = 'X';
                break;
        case FTW_SL:
                type = 'S';
                break;
        case FTW_SLN:
                type = 's';
                break;
        default:
                type = '?';
                break;
        }

        printf("[%c] %s\n", type, path);

        return (0);
}

int
main(int argc, char **argv)
{

        if (argc != 2) {
                printf("Usage %s <directory>\n", argv[0]);
                return (EX_USAGE);
        } else
                return (nftw(argv[1], nftw_callback, /* UNUSED */ 1, 0));
}
```

ERRORS

The **ftw**() and **nftw**() functions may fail and set *errno* for any of the errors specified for the library functions close(2), open(2), stat(2), malloc(3), opendir(3) and readdir(3). If the FTW_CHDIR flag is set, the **nftw**() function may fail and set *errno* for any of the errors specified for chdir(2). In addition, either function may fail and set *errno* as follows:

[EINVAL] The *maxfds* argument is less than 1.

SEE ALSO

chdir(2), close(2), open(2), stat(2), fts(3), malloc(3), opendir(3), readdir(3)

STANDARDS

The **ftw**() and **nftw**() functions conform to IEEE Std 1003.1-2001 ("POSIX.1").

HISTORY

These functions first appeared in AT&T System V Release 3 UNIX. Their first FreeBSD appearance was in FreeBSD 5.3.

BUGS

The *maxfds* argument is currently ignored.