NAME

ftw, nftw – walk a file tree

SYNOPSIS

#include <ftw.h>

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

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

MT-LEVEL

See the **NOTES** section of this page.

DESCRIPTION

ftw( ) recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, **ftw( )** calls the user-defined function *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a **stat** structure (see **stat**(2)) containing information about the object, and an integer. Possible values of the integer, defined in the **<ftw.h>** header, are:

FTW_F The object is a file.

FTW_D The object is a directory.

FTW_DNR The object is a directory that cannot be read. Descendants of the directory will not be processed.

FTW_NS **stat** failed on the object because of lack of appropriate permission or the object is a symbolic link that points to a non-existent file. The stat buffer passed to *fn* is undefined.

ftw( ) visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error is detected within **ftw( )** (such as an I/O error). If the tree is exhausted, **ftw( )** returns zero. If *fn* returns a nonzero value, **ftw( )** stops its tree traversal and returns whatever value was returned by *fn*.

The function **nftw( )** is similar to **ftw( )** except that it takes an additional argument, *flags*. The *flags* field is used to specify:

FTW_PHYS Physical walk, does not follow symbolic links. Otherwise, **nftw( )** will follow links but will not walk down any path that crosses itself.

FTW_MOUNT The walk will not cross a mount point.

FTW_DEPTH All subdirectories will be visited before the directory itself.

FTW_CHDIR The walk will change to each directory before reading it.

The function **nftw( )** calls *fn* with four arguments at each file and directory. The first argument is the pathname of the object, the second is a pointer to the **stat** buffer, the third is an integer giving additional information, and the fourth is a **struct FTW** that contains the following members:

```
int base;
int level;
```

base is the offset into the pathname of the base name of the object. **level** indicates the depth relative to the rest of the walk, where the root level is zero.

The values of the third argument are as follows:

FTW_F The object is a file.

FTW_D The object is a directory.

FTW_DP The object is a directory and subdirectories have been visited.

FTW_SL The object is a symbolic link.

FTW_SLN The object is a symbolic link that points to a non-existent file.

FTW_DNR The object is a directory that cannot be read. *fn* will not be called for any of its descendants.

FTW_NS **stat** failed on the object because of lack of appropriate permission. The stat buffer passed to *fn* is undefined. **stat** failure other than lack of appropriate permission. **EACCES** is considered an error and **nftw( )** will return –1.

Both **ftw( )** and **nftw( )** use one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *depth* must not be greater than the number of file descriptors currently available for use. **ftw( )** will run faster if *depth* is at least as large as the number of levels in the tree. When **ftw( )** and **nftw( )** return, they close any file descriptors they have opened; they do not close any file descriptors that may have been opened by *fn*.

RETURN VALUES

If successful, **ftw( )** and **nftw( )** return **0**. If either function detects an error other than **EACCES**, it returns –1, and sets the error type in **errno**.

SEE ALSO

stat(2), **longjmp**(3C), **malloc**(3C)

NOTES

Because **ftw( )** is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

ftw( ) uses **malloc**(3C) to allocate dynamic storage during its operation. If **ftw( )** is forcibly terminated, such as by **longjmp**(3C) being executed by *fn* or an interrupt routine, **ftw( )** will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

ftw( ) is safe in multi-thread applications. **nftw( )** is safe in multi-thread applications when the **FTW_CHDIR** flag is not set.