NAME

opendir − open a directory / readdir − read a directory

SYNOPSIS

```
#include <sys/types.h>

#include <dirent.h>

DIR *opendir(const char *name);

struct dirent *readdir(DIR *dir);
```

DESCRIPTION opendir

The **opendir()** function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

RETURN VALUE

The **opendir()** function returns a pointer to the directory stream or NULL if an error occurred.

DESCRIPTION readdir

The **readdir()** function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by *dir*. It returns NULL on reaching the end-of-file or if an error occurred.

The data returned by **readdir()** is overwritten by subsequent calls to **readdir()** for the same directory stream.

The *dirent* structure is defined as follows:

```
struct dirent {
        long            d_ino;              /* inode number */
        off_t           d_off;              /* offset to the next dirent */
        unsigned short  d_reclen;           /* length of this record */
        unsigned char   d_type;         /* type of file */
        char            d_name[256];    /* filename */
};
```

RETURN VALUE

The **readdir()** function returns a pointer to a dirent structure, or NULL if an error occurs or end-of-file is reached.

ERRORS

EACCES
Permission denied.

EMFILE
Too many file descriptors in use by process.

ENFILE
Too many files are currently open in the system.

ENOENT
Directory does not exist, or *name* is an empty string.

ENOMEM
Insufficient memory to complete the operation.

ENOTDIR
name is not a directory.

SEE ALSO

open(2), **readdir**(3), **closedir**(3), **rewinddir**(3), **seekdir**(3), **telldir**(3), **scandir**(3)

NAME

printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf − formatted output conversion

SYNOPSIS

```
#include <stdio.h>

int printf(const char * format, ...);
int fprintf(FILE *stream, const char * format, ...);
int sprintf(char *str, const char * format, ...);
int snprintf(char *str, size_t size, const char * format, ...);
 ...
```

DESCRIPTION

The functions in the **printf**() family produce output according to a *format* as described below. The function **printf**() writes output to *stdout*, the standard output stream; **fprintf**() writes output to the given output *stream*; **sprintf**() and **snprintf**(), write to the character string *str*.

The function **snprintf**() writes at most *size* bytes (including the trailing null byte ('\0')) to *str*.

These functions write the output under the control of a *format* string that specifies how subsequent arguments are converted for output.

Return value

Upon successful return, these functions return the number of characters printed (not including the trailing '\0' used to end output to strings).

If an output error is encountered, a negative value is returned.

Format of the format string

The format string is a character string, beginning and ending in its initial shift state, if any. The format string is composed of zero or more directives: ordinary characters (not **%**), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character **%**, and ends with a *conversion specifier*. In between there may be (in this order) zero or more *flags*, an optional minimum *field width*, an optional *precision* and an optional *length modifier*.

The conversion specifier

A character that specifies the type of conversion to be applied. An example for a conversion specifier is:

d, **i** The *int* argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1.

o, **u**, **x**, **X**
The *unsigned int* argument is converted to unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal (**x** and **X**) notation.

c The *int* argument is converted to an *unsigned char* , and the resulting character is written.

s The *const char ** argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating null byte ('\0'); if a precision is specified, no more than the number specified are written.

SEE ALSO

printf(1), **asprintf**(3), **dprintf**(3), **scanf**(3), **setlocale**(3), **wcrtomb**(3), **wprintf**(3), **locale**(5)

NAME

stat, lstat − get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *path, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

DESCRIPTION

These functions return information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file.

stat stats the file pointed to by *path* and fills in *buf*.

lstat is identical to **stat**, except in the case of a symbolic link, where the link itself is stat-ed, not the file that it refers to.

They all return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t       st_dev;      /* device */
    ino_t       st_ino;      /* inode */
    mode_t      st_mode;     /* protection */
    nlink_t     st_nlink;    /* number of hard links */
    uid_t       st_uid;      /* user ID of owner */
    gid_t       st_gid;      /* group ID of owner */
    dev_t       st_rdev;     /* device type (if inode device) */
    off_t       st_size;     /* total size, in bytes */
    blksize_t   st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t    st_blocks;   /* number of blocks allocated */
    time_t      st_atime;    /* time of last access */
    time_t      st_mtime;    /* time of last modification */
    time_t      st_ctime;    /* time of last status change */
};
```

The value *st_size* gives the size of the file (if it is a regular file or a symlink) in bytes. The size of a symlink is the length of the pathname it contains, without trailing NUL.

The following POSIX macros are defined to check the file type in the field *st_mode:*

S_ISREG(m)	is it a regular file?
S_ISDIR(m)	directory?
S_ISLNK(m)	symbolic link?

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

ERRORS

EACCES	Search permisson is denied for one of the directories in the path prefix of *path.*
ENOENT	A component of *path* does not exist, or *path* is an empty string.
ENOTDIR	A component of the path prefix of *path* is not a directory.