

**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**

strcat, strcat – concatenate two strings

**SYNOPSIS**

```
#include <string.h>
```

```
char *strcat(char *dest, const char *src);
```

```
char *strncat(char *dest, const char *src, size_t n);
```

**DESCRIPTION**

The **strcat()** function appends the *src* string to the *dest* string, overwriting the null byte ('\0') at the end of *dest*, and then adds a terminating null byte. The strings may not overlap, and the *dest* string must have enough space for the result.

The **strncat()** function is similar, except that

\* it will use at most *n* characters from *src*; and

\* *src* does not need to be null terminated if it contains *n* or more characters.

As with **strcat()**, the resulting string in *dest* is always null terminated.

If *src* contains *n* or more characters, **strncat()** writes *n+1* characters to *dest* (*n* from *src* plus the terminating null byte). Therefore, the size of *dest* must be at least *strlen(dest)+n+1*.

A simple implementation of **strncat()** might be:

```
char*
strncat(char *dest, const char *src, size_t n)
{
    size_t dest_len = strlen(dest);
    size_t i;

    for (i = 0 ; i < n && src[i] != '\0' ; i++)
        dest[dest_len + i] = src[i];
    dest[dest_len + i] = '\0';

    return dest;
}
```

**RETURN VALUE**

The **strcat()** and **strncat()** functions return a pointer to the resulting string *dest*.

**CONFORMING TO**

SVr4, 4.3BSD, C89, C99.

**SEE ALSO**

**bcopy(3)**, **memcpy(3)**, **memccpy(3)**, **strcpy(3)**, **strncpy(3)**, **wscat(3)**, **wcsncat(3)**

**COLOPHON**

This page is part of release 3.05 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at <http://www.kernel.org/doc/man-pages/>.

**NAME**

stat, lstat – get file status

**SYNOPSIS**

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

```
int stat(const char *file_name, struct stat *buf);
int lstat(const char *file_name, 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 *file\_name* 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.

Not all of the Linux filesystems implement all of the time fields. Some file system types allow mounting in such a way that file accesses do not cause an update of the *st\_atime* field. (See ‘noatime’ in **mount(8)**.)

The field *st\_atime* is changed by file accesses, e.g. by **execve(2)**, **mknod(2)**, **pipe(2)**, **utime(2)** and **read(2)** (of more than zero bytes). Other routines, like **mmap(2)**, may or may not update *st\_atime*.

The field *st\_mtime* is changed by file modifications, e.g. by **mknod(2)**, **truncate(2)**, **utime(2)** and **write(2)** (of more than zero bytes). Moreover, *st\_mtime* of a directory is changed by the creation or deletion of files in that directory. The *st\_mtime* field is *not* changed for changes in owner, group, hard link count, or mode.

The field *st\_ctime* is changed by writing or by setting inode information (i.e., owner, group, link count, mode, etc.).

**RETURN VALUE**

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