NAME

opendir – open a directory / readdir – read a directory / closedir – close a directory

SYNOPSIS

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

#include <dirent.h>

DIR *opendir(const char *name);

int closedir(DIR *dir);

struct dirent *readdir(DIR *dir);
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);
```

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 closedir

The **closedir**() function closes the directory stream associated with *dir*. The directory stream descriptor *dir* is not available after this call.

RETURN VALUE

The **closedir()** function returns 0 on success. On error, −1 is returned, and *errno* is set appropriately.

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; not supported by all filesystem types */
    char            d_name[256]; /* filename */
};
```

RETURN VALUE

On success, **readdir()** returns a pointer to a dirent structure. If the end of the directory stream is reached, NULL is returned and errno is not changed. If an error occurs, NULL is returned and errno is set appropriately.

ERRORS

EACCES

Permission denied.

ENOENT

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

ENOTDIR

name is not a directory.

NAME

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

DESCRIPTION

These functions return information about a file. No permissions are required on the file itself, but — in the case of **stat**() and **lstat**() — execute (search) permission is required on all of the directories in *path* that lead to the file.

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

lstat() is identical to **stat**(), except that if *path* is a symbolic link, then the link itself is stat-ed, not the file that it refers to.

fstat() is identical to **stat**(), except that the file to be stat-ed is specified by the file descriptor *fd*.

All of these system calls return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    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 ID (if special file) */
    unsigned  st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system 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 *st_dev* field describes the device on which this file resides.

The *st_rdev* field describes the device that this file (inode) represents.

The *st_size* field gives the size of the file (if it is a regular file or a symbolic link) in bytes. The size of a symlink is the length of the pathname it contains, without a trailing null byte.

The *st_blocks* field indicates the number of blocks allocated to the file, 512-byte units. (This may be smaller than *st_size*/512 when the file has holes.)

The *st_blksize* field gives the "preferred" blocksize for efficient file system I/O. (Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite.)

Not all of the Linux file systems 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.

The field *st_atime* is changed by file accesses, for example, 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, for example, 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.).

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

S_ISREG(m)	is it a regular file?
S_ISDIR(m)	directory?
S_ISCHR(m)	character device?
S_ISBLK(m)	block device?
S_ISFIFO(m)	FIFO (named pipe)?
S_ISLNK(m)	symbolic link? (Not in POSIX.1-1996.)
S_ISSOCK(m)	socket? (Not in POSIX.1-1996.)

RETURN VALUE

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

ERRORS

EACCES
Search permission is denied for one of the directories in the path prefix of *path*.

EBADF
fd is bad.

EFAULT
Bad address.

ELOOP
Too many symbolic links encountered while traversing the path.

ENAMETOOLONG
File name too long.

ENOENT
A component of the path *path* does not exist, or the path is an empty string.

ENOMEM
Out of memory (i.e., kernel memory).

ENOTDIR
A component of the path is not a directory.

NAME

strcat, strchr, strcmp, strcpy, strdup, strlen, strncat, strncmp, strncpy, strstr, strtok − string operations

SYNOPSIS

#include <string.h>

char *strcat(char *_dest_, const char *_src_);
Append the string *src* to the string *dest*, returning a pointer *dest*.

char *strchr(const char *_s_, int _c_);
Return a pointer to the first occurrence of the character *c* in the string *s*.

int strcmp(const char *_s1_, const char *_s2_);
Compare the strings *s1* with *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

char *strcpy(char *_dest_, const char *_src_);
Copy the string *src* to *dest*, returning a pointer to the start of *dest*.

char *strdup(const char *_s_);
Return a duplicate of the string *s* in memory allocated using **malloc**(3).

size_t strlen(const char *_s_);
Return the length of the string *s*.

char *strncat(char *_dest_, const char *_src_, size_t _n_);
Append at most *n* characters from the string *src* to the string *dest*, returning a pointer to *dest*.

int strncmp(const char *_s1_, const char *_s2_, size_t _n_);
Compare at most *n* bytes of the strings *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

char *strncpy(char *_dest_, const char *_src_, size_t _n_);
Copy at most *n* bytes from string *src* to *dest*, returning a pointer to the start of *dest*.

char *strstr(const char *_haystack_, const char *_needle_);
Find the first occurrence of the substring *needle* in the string *haystack*, returning a pointer to the found substring.

char *strtok(char *_s_, const char *_delim_);
Extract tokens from the string *s* that are delimited by one of the bytes in *delim*.

DESCRIPTION

The string functions perform operations on null-terminated strings.

NAME

time – get time in seconds

SYNOPSIS

#include <time.h>

time_t time(time_t *_tloc_);

DESCRIPTION

time() returns the time as the number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).

The *tloc* argument is obsolescent and should always be NULL in new code. When *tloc* is NULL, the call cannot fail.

RETURN VALUE

On success, the value of time in seconds since the Epoch is returned. On error, *((time_t) −1)* is returned, and *errno* is set to indicate the error.

ERRORS

EFAULT
tloc points outside your accessible address space.

NAME

unlink – delete file

SYNOPSIS

#include <unistd.h>

int unlink(const char *_path_);

DESCRIPTION

unlink() deletes a name from the filesystem. If that name was the last link to a file and no processes have the file open, the file is deleted and the space it was using is made available for reuse.

If the name was the last link to a file but any processes still have the file open, the file will remain in existence until the last file descriptor referring to it is closed.

RETURN VALUE

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

ERRORS

EACCES
Permission denied.

EFAULT
pathname points outside your accessible address space.

EIO An I/O error occurred.

EISDIR
pathname refers to a directory. (This is the non-POSIX value returned since Linux 2.1.132.)

ELOOP
Too many symbolic links were encountered in translating *pathname*.

ENAMETOOLONG
pathname was too long.

ENOENT
A component in *pathname* does not exist or is a dangling symbolic link, or *pathname* is empty.

ENOMEM
Insufficient kernel memory was available.

ENOTDIR
A component used as a directory in *pathname* is not, in fact, a directory.

EPERM
The filesystem does not allow unlinking of files.

EROFS
pathname refers to a file on a read-only filesystem.