

**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);
```

Feature Test Macro Requirements for glibc (see **feature\_test\_macros(7)**):

```
lstat(): _BSD_SOURCE || _XOPEN_SOURCE >= 500
```

**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) */
    off_t    st_size;   /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t st_blocks; /* number of 512B 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 **major(3)** and **minor(3)** macros may be useful to decompose the device ID in this field.)

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 and/or directory accesses do not cause an update of the *st\_atime* field. (See *noatime*, *nodiratime*, and *relatime* in **mount**(8), and related information in **mount**(2).) In addition, *st\_atime* is not updated if a file is opened with the **O\_NOATIME**; see **open**(2).

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:

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

The following flags are defined for the *st\_mode* field:

|                 |         |  |
|-----------------|---------|--|
| <b>S_IFMT</b>   | 0170000 | bit mask for the file type bit fields          |
| <b>S_IFSOCK</b> | 0140000 | socket   |
| <b>S_IFLNK</b>  | 0120000 | symbolic link                                  |
| <b>S_IFREG</b>  | 0100000 | regular file                                   |
| <b>S_IFBLK</b>  | 0060000 | block device                                   |
| <b>S_IFDIR</b>  | 0040000 | directory                                      |
| <b>S_IFCHR</b>  | 0020000 | character device                               |
| <b>S_IFIFO</b>  | 0010000 | FIFO   |
| <b>S_ISUID</b>  | 0004000 | set UID bit                                    |
| <b>S_ISGID</b>  | 0002000 | set-group-ID bit (see below)                   |
| <b>S_ISVTX</b>  | 0001000 | sticky bit (see below)                         |
| <b>S_IRWXU</b>  | 00700   | mask for file owner permissions                |
| <b>S_IRUSR</b>  | 00400   | owner has read permission                      |
| <b>S_IWUSR</b>  | 00200   | owner has write permission                     |
| <b>S_IXUSR</b>  | 00100   | owner has execute permission                   |
| <b>S_IRWXG</b>  | 00070   | mask for group permissions                     |
| <b>S_IRGRP</b>  | 00040   | group has read permission                      |
| <b>S_IWGRP</b>  | 00020   | group has write permission                     |
| <b>S_IXGRP</b>  | 00010   | group has execute permission                   |
| <b>S_IRWXO</b>  | 00007   | mask for permissions for others (not in group) |
| <b>S_IROTH</b>  | 00004   | others have read permission                    |
| <b>S_IWOTH</b>  | 00002   | others have write permission                   |
| <b>S_IXOTH</b>  | 00001   | others have execute permission                 |

The set-group-ID bit (**S\_ISGID**) has several special uses. For a directory it indicates that BSD semantics is to be used for that directory: files created there inherit their group ID from the directory, not from the

effective group ID of the creating process, and directories created there will also get the **S\_ISGID** bit set. For a file that does not have the group execution bit (**S\_IXGRP**) set, the set-group-ID bit indicates mandatory file/record locking.

The sticky bit (**S\_ISVTX**) on a directory means that a file in that directory can be renamed or deleted only by the owner of the file, by the owner of the directory, and by a privileged process.

## 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*. (See also **path\_resolution(7)**.)

### 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 *path* does not exist, or *path* is an empty string.

### ENOMEM

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

### ENOTDIR

A component of the path prefix of *path* is not a directory.

### EOVERFLOW

(**stat()**) *path* refers to a file whose size cannot be represented in the type *off\_t*. This can occur when an application compiled on a 32-bit platform without *-D\_FILE\_OFFSET\_BITS=64* calls **stat()** on a file whose size exceeds  $(2^{31}-1)$  bits.

## CONFORMING TO

These system calls conform to SVr4, 4.3BSD, POSIX.1-2001.

Use of the *st\_blocks* and *st\_blksize* fields may be less portable. (They were introduced in BSD. The interpretation differs between systems, and possibly on a single system when NFS mounts are involved.)

POSIX does not describe the **S\_IFMT**, **S\_IFSOCK**, **S\_IFLNK**, **S\_IFREG**, **S\_IFBLK**, **S\_IFDIR**, **S\_IFCHR**, **S\_IFIFO**, **S\_ISVTX** bits, but instead demands the use of the macros **S\_ISDIR()**, etc. The **S\_ISLNK()** and **S\_ISSOCK()** macros are not in POSIX.1-1996, but both are present in POSIX.1-2001; the former is from SVID 4, the latter from SUSv2.

Unix V7 (and later systems) had **S\_IREAD**, **S\_IWRITE**, **S\_IEXEC**, where POSIX prescribes the synonyms **S\_IRUSR**, **S\_IWUSR**, **S\_IXUSR**.

## Other Systems

Values that have been (or are) in use on various systems:

| hex  | name   | ls | octal  | description  |
|------|--------|----|--------|--|
| f000 | S_IFMT |    | 170000 | mask for file type   |
| 0000 |        |    | 000000 | SCO out-of-service inode; BSD unknown type; SVID-v2 and XPG2 have both 0 and 0100000 for ordinary file |

|      |          |    |        |   |
|------|----------|----|--------|---|
| 1000 | S_IFIFO  | p  | 010000 | FIFO (named pipe)   |
| 2000 | S_IFCHR  | c  | 020000 | character special (V7)  |
| 3000 | S_IFMPC  |    | 030000 | multiplexed character special (V7)  |
| 4000 | S_IFDIR  | d/ | 040000 | directory (V7)  |
| 5000 | S_IFNAM  |    | 050000 | XENIX named special file<br>with two subtypes, distinguished by<br><i>st_rdev</i> values 1, 2   |
| 0001 | S_INSEM  | s  | 000001 | XENIX semaphore subtype of IFNAM  |
| 0002 | S_INSHD  | m  | 000002 | XENIX shared data subtype of IFNAM  |
| 6000 | S_IFBLK  | b  | 060000 | block special (V7)  |
| 7000 | S_IFMPB  |    | 070000 | multiplexed block special (V7)  |
| 8000 | S_IFREG  | -  | 100000 | regular (V7)  |
| 9000 | S_IFCMP  |    | 110000 | VxFS compressed   |
| 9000 | S_IFNWK  | n  | 110000 | network special (HP-UX)   |
| a000 | S_IFLNK  | l@ | 120000 | symbolic link (BSD)   |
| b000 | S_IFSHAD |    | 130000 | Solaris shadow inode for ACL<br>(not seen by userspace)   |
| c000 | S_IFSOCK | s= | 140000 | socket (BSD; also "S_IFSOC" on VxFS)  |
| d000 | S_IFDOOR | D> | 150000 | Solaris door  |
| e000 | S_IFWHT  | w% | 160000 | BSD whiteout (not used for inode)   |
| 0200 | S_ISVTX  |    | 001000 | sticky bit: save swapped text even<br>after use (V7)<br>reserved (SVID-v2)<br>On non-directories: don't cache this<br>file (SunOS)<br>On directories: restricted deletion<br>flag (SVID-v4.2) |
| 0400 | S_ISGID  |    | 002000 | set-group-ID on execution (V7)<br>for directories: use BSD semantics for<br>propagation of GID  |
| 0400 | S_ENFMT  |    | 002000 | System V file locking enforcement (shared<br>with S_ISGID)  |
| 0800 | S_ISUID  |    | 004000 | set-user-ID on execution (V7)   |
| 0800 | S_CDF    |    | 004000 | directory is a context dependent<br>file (HP-UX)  |

A sticky command appeared in Version 32V AT&T UNIX.

## NOTES

Since kernel 2.5.48, the *stat* structure supports nanosecond resolution for the three file timestamp fields. Glibc exposes the nanosecond component of each field using names either of the form *st\_atim.tv\_nsec*, if the **\_BSD\_SOURCE** or **\_SVID\_SOURCE** feature test macro is defined, or of the form *st\_atimensec*, if neither of these macros is defined. On file systems that do not support sub-second timestamps, these nanosecond fields are returned with the value 0.

On Linux, **lstat()** will generally not trigger automounter action, whereas **stat()** will.

For most files under the */proc* directory, **stat()** does not return the file size in the *st\_size* field; instead the field is returned with the value 0.

## Underlying kernel interface

Over time, increases in the size of the *stat* structure have led to three successive versions of **stat()**: *sys\_stat()* (slot *\_\_NR\_oldstat*), *sys\_newstat()* (slot *\_\_NR\_stat*), and *sys\_stat64()* (new in kernel 2.4; slot *\_\_NR\_stat64*). The glibc **stat()** wrapper function hides these details from applications, invoking the most recent version of the system call provided by the kernel, and repacking the returned information if required for old binaries. Similar remarks apply for **fstat()** and **lstat()**.

**EXAMPLE**

The following program calls **stat()** and displays selected fields in the returned *stat* structure.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    struct stat sb;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s <pathname>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if (stat(argv[1], &sb) == -1) {
        perror("stat");
        exit(EXIT_SUCCESS);
    }

    printf("File type:          ");

    switch (sb.st_mode & S_IFMT) {
    case S_IFBLK: printf("block device\n");      break;
    case S_IFCHR: printf("character device\n");    break;
    case S_IFDIR: printf("directory\n");           break;
    case S_IFIFO: printf("FIFO/pipe\n");           break;
    case S_IFLNK: printf("symlink\n");             break;
    case S_IFREG: printf("regular file\n");         break;
    case S_IFSOCK: printf("socket\n");             break;
    default:      printf("unknown?\n");            break;
    }

    printf("I-node number:      %ld\n", (long) sb.st_ino);

    printf("Mode:                %lo (octal)\n",
           (unsigned long) sb.st_mode);

    printf("Link count:          %ld\n", (long) sb.st_nlink);
    printf("Ownership:          UID=%ld  GID=%ld\n",
           (long) sb.st_uid, (long) sb.st_gid);

    printf("Preferred I/O block size: %ld bytes\n",
           (long) sb.st_blksize);
    printf("File size:             %lld bytes\n",
           (long long) sb.st_size);
    printf("Blocks allocated:       %lld\n",
           (long long) sb.st_blocks);

    printf("Last status change:     %s", ctime(&sb.st_ctime));
}
```

```
printf("Last file access:      %s", ctime(&sb.st_atime));  
printf("Last file modification: %s", ctime(&sb.st_mtime));  
  
exit(EXIT_SUCCESS);  
}
```

**SEE ALSO**

**access(2), chmod(2), chown(2), fstatat(2), readlink(2), utime(2), capabilities(7), symlink(7)**

**COLOPHON**

This page is part of release 3.22 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/>.