NAME

stat, fstat, 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 fstat(int *filedes***, 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.

fstat is identical to **stat**, only the open file pointed to by *filedes* (as returned by **open**(2)) is stat-ed in place of *file_name*.

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 value *st_blocks* gives the size of the file in 512-byte blocks. (This may be smaller than *st_size*/512 e.g. when the file has holes.) The value *st_blksize* 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 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.).

The following POSIX macros are defined to check the file type:

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?
S_ISLNK(m)	symbolic link? (Not in POSIX.1-1996.)
S_ISSOCK(m)	socket? (Not in POSIX.1-1996.)

The following flags are defined for the *st_mode* field:

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

The set GID 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, it 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.

SEE ALSO

chmod(2), **chown**(2), **readlink**(2), **utime**(2), **capabilities**(7)