Current stat / fstat / lstat Linux Man Page

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 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 gid 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 root.

RETURN VALUE

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

ERRORS

EBADF
filedes is bad.
ENOENT
A component of the path *file_name* does not exist, or the path is an empty string.
ENOTDIR
A component of the path is not a directory.
ELOOP
Too many symbolic links encountered while traversing the path.
EFAULT
Bad address.
EACCES
Permission denied.
ENOMEM
Out of memory (i.e. kernel memory).
ENAMETOOLONG
File name too long.

CONFORMING TO

The **stat** and **fstat** calls conform to SVr4, SVID, POSIX, X/OPEN, BSD 4.3. The **lstat** call conforms to 4.3BSD and SVr4. SVr4 documents additional **fstat** error conditions EINTR, ENOLINK, and EOVERFLOW. SVr4 documents additional **stat** and **lstat** error conditions EACCES, EINTR, EMULTIHOP, ENOLINK, and EOVERFLOW. Use of the *st_blocks* and *st_blksize* fields may be less portable. (They were introduced in BSD. Are not specified by POSIX. 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 will be in the next POSIX standard; the former is from SVID 4v2, 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
				with two subtypes, distinguished by st_rdev 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 (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 after use (V7)
				reserved (SVID-v2)
				On non-directories: don't cache this file (SunOS)
				On directories: restricted deletion flag (SVID-v4.2)
0400	S_ISGID		002000	set group ID on execution (V7)
				for directories: use BSD semantics for propagation of gid
0400	S_ENFMT		002000	SysV file locking enforcement (shared w/ S_ISGID)
0800	S_ISUID		004000	set user ID on execution (V7)
0800	S_CDF		004000	directory is a context dependent file (HP-UX)

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

SEE ALSO

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

Proposed statlite / fstatlite / lstatlite Linux Man Page

NAME

statlite, fstatlite, lstatlite - get file status

SYNOPSIS

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

int statlite(const char *file_name, struct statlite *buf);
int fstatlite(int filedes, struct statlite *buf);
int lstatlite(const char *file_name, struct statlite *buf);
```

DESCRIPTION

These functions return some manditory and possibly some optional information 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. This family of stat calls, the lite family, is provided to allow for file I/O performance not to be comprimised by frequent use of stat information lookup and for lower overhead for stat operations in general.

statlite stats the file pointed to by *file_name* and fills in *buf*.

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

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

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

```
struct statlite {
    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)*/
unsigned long     st_litemask; /* bit mask for optional field
accuracy */
/* The st-litemask field can be used as an input parameter to
specify which of the optional fields below are required to be
guaranteed to be accurate at the time of the call.  Setting the
mask bit associated with the optional field to a 1 value requires
the associated optional field to be returned accurately.  Setting
the mask bit associated with the optional field to a 0 value
allows the file system to return an unreliable value in that
optional field.
```

```
        In all cases, the st_litemask field will be returned with the
        accuracy information for every optional field.  A value of 1
        means the field is accurate; a value of 0 means the value may not
        be accurate.  If all the call is made with st_litemask as all
        zeros directs the file system to do a low overhead stat operation
        and fill in optional fields as appropriate and accuracy bits in
        the st_litemask as appropriate for that file systems version of a
        low overhead stat operation.
            /* Fields below here are optionally provided and are
          guaranteed to be correct only if there corresponding bit is set
        to 1 in the manditory st_litemask field, with the lite versions
        of the stat family of calls  */
            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 change */

            /* End of optional fields  */

};
```

The following POSIX macros are defined to check to see if an optional field is accurate, a 1 means the field is accurate a 0 means accuracy is not guaranteed for that field:

```
SLITE_SIZET(m)
SLITE_BLKSIZE(m)
SLITE_BLOCKS(m)
SLITE_ATIME(m)
SLITE_MTIME(m)
SLITE_CTIME(m)
```

The following POSIX macros are defined to set the accuracy bit in the st_litemask field for the corresponding optional field.

```
S_SLITE_SIZET(m)
S_SLITE_BLKSIZE(m)
S_SLITE_BLOCKS(m)
S_SLITE_ATIME(m)
S_SLITE_MTIME(m)
S_SLITE_CTIME(m)
```

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 gid 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 root.

RETURN VALUE

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

ERRORS

EBADF
filedes is bad.
ENOENT
A component of the path *file_name* does not exist, or the path is an empty string.
ENOTDIR
A component of the path is not a directory.
ELOOP
Too many symbolic links encountered while traversing the path.
EFAULT
Bad address.
EACCES
Permission denied.
ENOMEM
Out of memory (i.e. kernel memory).
ENAMETOOLONG
File name too long.

CONFORMING TO

The **statlite** and **fstatlite** calls conform to SVr4, SVID, POSIX, X/OPEN, BSD 4.3, with the exception of the optional fields. The **lstatlite** call conforms to 4.3BSD and SVr4 with the exception of the optional fields. SVr4 documents additional error conditions EINTR, ENOLINK, and EOVERFLOW in **fstat** that are relevant for **fstatlite**. SVr4 documents additional error conditions EACCES, EINTR, EMULTIHOP, ENOLINK, and EOVERFLOW in **stat** and **lstat** that are relevant for **statlite** and **lstatlite**. Use of the *st_blocks* and *st_blksize* fields may be less portable. (They were introduced in BSD. Are not specified by POSIX. 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 will be in the next POSIX standard; the former is from SVID 4v2, 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
				with two subtypes, distinguished by st_rdev 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 (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 after use (V7)
				reserved (SVID-v2)
				On non-directories: don't cache this file (SunOS)
				On directories: restricted deletion flag (SVID-v4.2)
0400	S_ISGID		002000	set group ID on execution (V7)
				for directories: use BSD semantics for propagation of gid
0400	S_ENFMT		002000	SysV file locking enforcement (shared w/ S_ISGID)
0800	S_ISUID		004000	set user ID on execution (V7)
0800	S_CDF		004000	directory is a context dependent file (HP-UX)

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

SEE ALSO

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