stat(2) — Linux manual page

NAME top

stat, fstat, lstat, fstatat - получить статус файла

СИНОПСИС top

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

int stat(const char *restrict pathname,
         struct stat *restrict statbuf);
int fstat(int fd, struct stat *statbuf);
int lstat(const char *restrict pathname,
         struct stat *restrict statbuf);

#include <fcntl.h>/* Определение констант AT_* */
#include <sys/stat.h>

int fstatat(int dirfd, const char *restrict pathname,
         struct stat *restrict statbuf, int flags);
```

Требования к макросу Feature Test для glibc (см.
feature_test_macros(7)):

lstat():
/* Начиная с glibc 2.20 */ _DEFAULT_SOURCE
|| _XOPEN_SOURCE >= 500
|| /* Начиная с glibc 2.10: */ _POSIX_C_SOURCE >= 200112L
|| /* Glibc 2.19 и более ранние версии */ _BSD_SOURCE

fstatat():
Начиная с glibc 2.10:
_POSIX_C_SOURCE >= 200809L
До glibc 2.10:
_ATFILE_SOURCE

ОПИСАНИЕ top

Эти функции возвращают информацию о файле в буфере
, на который указывает *statbuf*. Для самого файла не требуется никаких разрешений
, но – в случае **stat**(), **fstatat**() и **lstat**() – execute
разрешение (поиск) требуется для всех каталогов в
pathname, которые ведут к файлу.

stat() и **fstatat**() извлекают информацию о файле, на который указывает
путь; различия для **fstatat**() описаны
ниже.

lstat() идентичен **stat**(), за исключением того, что if *pathname* если это
символическая ссылка, то она возвращает информацию о самой ссылке,
а не о файле, на который ссылается ссылка.

функция fstat() идентична **функции stat**(), за исключением того, что файл
, информация о котором должна быть получена, задается файловым
дескриптором *fd*.

Структура stat

Все эти системные вызовы возвращают *stat* структура, которая содержит следующие поля:

```
struct stat {
dev_t st_dev; /* Идентификатор устройства, содержащего файл */
ino_t st_ino; /* Номер индекса */
mode_t st_mode; /* Тип и режим файла */
nlink_t st_nlink; /* Количество жестких ссылок */
uid_t st_uid; /* Идентификатор пользователя владельца */
gid_t st_gid; /* Идентификатор группы владельца */
dev_t st_rdev; /* Идентификатор устройства (если специальный файл) */
off_t st_size; /* Общий размер, в байтах */
blksize_t st_blksize; /* Размер блока для ввода-вывода файловой системы */
blkcnt_t st_blocks; /* Количество выделенных блоков 512B */

/* Начиная с Linux 2.6, ядро поддерживает наносекундную
точность для следующих полей временных меток.
Подробные сведения до Linux 2.6 см. В ПРИМЕЧАНИЯХ. */

struct timespec st_atim; /* Время последнего доступа */
struct timespec st_mtim; /* Время последней модификации */
struct timespec st_ctim; /* Время последнего изменения статуса */

#define st_atime st_atim.tv_sec /* Обратная совместимость */
#определить st_mtime st_mtim.tv_sec
#определить st_ctime st_ctim.tv_sec
};
```

*Порядок*полей в структуре *stat* несколько различается в разных архитектурах. Кроме того, приведенное выше определение не показывает байты заполнения, которые могут присутствовать между некоторыми полями в различных архитектурах.
Если вам нужно узнать подробности, обратитесь к glibc и исходному коду ядра.

Примечание: по соображениям производительности и простоты разные поля в структуре *stat* могут содержать информацию о состоянии из разных моментов во время выполнения системного вызова. Например, если *st_mode* или *st_uid* изменены другим процессом путем вызова chmod(2) или chown(2), **stat**() может вернуть старый *st_mode* вместе с новым *st_uid*или старый *st_uid* вместе с новым *st_mode*.

Поля в структуре *stat* следующие:

st_dev Это поле описывает устройство, на котором находится этот файл. (Основные (3) и для разложения идентификатора устройства в этом поле могут быть полезны второстепенные макросы(3).)

st_ino Это поле содержит номер индекса файла.

st_mode
Это поле содержит тип файла и режим. См. inode(7) для получения дополнительной информации.

st_nlink
Это поле содержит количество жестких ссылок на файл.

st_uid Это поле содержит идентификатор пользователя владельца файла.

st_gid Это поле содержит идентификатор владельца группы файла.

st_rdev
Это поле описывает устройство, на котором находится этот файл (inode).
представляет.

st_size
В этом поле указывается размер файла (если это обычный файл или символическая ссылка) в байтах. Размер символьной ссылки – это длина пути, который она содержит, без

завершающего нулевого байта.

st_blksize
В этом поле указывается "предпочтительный" размер блока для эффективного ввода-вывода файловой системы.

st_blocks
В этом поле указывается количество блоков, выделенных для файла, в 512-байтовых единицах. (Это может быть меньше, чем *st_size*/ 512, если файл имеет отверстия.)

st_atime
Это время последнего доступа к данным файла.

st_mtime
Это время последней модификации данных файла.

st_ctime
Это метка времени последнего изменения статуса файла (время последнего изменения inode).

Для получения дополнительной информации о вышеуказанных полях см. inode(7).

fstatat()
fstatatсистемный вызов () - это более общий интерфейс для доступа к информации о файле, который все еще может обеспечить точное поведение каждого из **stat**(), **lstat**() и **fstat**() .

Если путь, указанный в *pathname*, является относительным, то он интерпретируется относительно каталога, на который ссылается файловый дескриптор *dirfd* (а не относительно текущего рабочего каталога вызывающего процесса, как это делается **stat**() и **lstat**() для относительного пути).

Если *pathname* является относительным, а *dirfd* - специальным значением **AT_FDCWD**,

то *pathname* интерпретируется относительно текущего рабочего каталога вызывающего процесса (например, **stat**() и **lstat**()).

Если *pathname* является абсолютным, то *dirfd* игнорируется.

флаги могут быть либо 0, либо включать один или несколько из следующих флагов ORed:

AT_EMPTY_PATH (начиная с Linux 2.6.39)
Если *путь* является пустой строкой, работайте с файлом , на который ссылается *dirfd* (который, возможно, был получен с помощью флага open(2) O_PATH). В этом случае *dirfd* может ссылаться на любой тип файла, а не только на каталог, и поведение **fstatat**() аналогично поведению **fstat**() . Если *значение dirfd* равно **AT_FDCWD**, вызов выполняется в текущем рабочем каталоге. Этот флаг специфичен для Linux; определите **_GNU_SOURCE**, чтобы получить его определение.

AT_NO_AUTOMOUNT (начиная с Linux 2.6.38)
Не устанавливайте автоматически компонент terminal ("basename") *в pathname* если это каталог, который является точкой автоматической установки.
Это позволяет вызывающему абоненту собирать атрибуты точки автоматической установки (а не местоположения, которое он будет монтировать).
Начиная с Linux 4.14, также не создавайте экземпляр несуществующего имени в каталоге по требованию, например, используемом для косвенных карт automounter. Этот флаг не действует, если точка монтирования уже смонтирована.

Оба **stat**() и **lstat**() действуют так, как будто **AT_NO_AUTOMOUNT** был установлен.

AT_NO_AUTOMOUNT может использоваться в инструментах, которые сканируют каталоги, чтобы предотвратить массовый автоматический монтаж каталога точек автоматического монтажа.

Этот флаг специфичен для Linux; определите **_GNU_SOURCE**, чтобы получить его определение.

AT_SYMLINK_NOFOLLOW
Если *pathname* является символической ссылкой, не разыменовывайте ее: вместо этого возвращайте информацию о самой ссылке, например **lstat**(). (По умолчанию **fstatat**() разыменовывает символические ссылки, такие как **stat**() .)

См. openat(2) для объяснения необходимости **fstatat**() .

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ top

При успешном выполнении возвращается ноль. При ошибке возвращается значение -1, а значение errno указывает на ошибку.

ОШИБКИ сверху

Разрешение на поиск EACCES отказано для одного из каталогов в префиксе пути *pathname*. (См. Также path_resolution(7).)

EBADF *fd* не является допустимым открытым файловым дескриптором.

Путь EBADF (**fstatat**()) относителен, *но dirfd* не является ни **AT_FDCWD**, ни допустимым файловым дескриптором.

EFAULT Плохой адрес.

EINVAL (**fstatat**()) Недопустимый флаг, указанный в *flags*.

ELOOP Слишком много символических ссылок, встречающихся при пересечении пути.

Путь ENAMETOOLONG слишком длинный.

ENOENT Компонент *pathname* не существует или является висячей символической ссылкой.

ENOENT *pathname* - это пустая строка, и **AT_EMPTY_PATH** не был указан во *флагах*.

ENOMEM из памяти (т.е. Памяти ядра).

ENOTDIR
Компонент префикса пути *pathname* не является каталогом.

ENOTDIR
(**fstatat**()) *путь* является относительным, а *dirfd* - это файловый дескриптор, ссылающийся на файл, отличный от каталога.

Путь к EOVERFLOW или *fd* относится к файлу, размер, номер индекса или количество блоков которого не могут быть представлены соответственно типами *off_t*, *ino_t*или *blkcnt_t*. Эта ошибка может возникнуть, например, когда приложение, скомпилированное на 32-разрядной платформе, не *вызывает -D_FILE_OFFSET_BITS=64* **stat**() для файла, размер которого превышает *(1<<31) -1* байт.

ВЕРСИИ top

fstatat() был добавлен в Linux в ядре 2.6.16; поддержка библиотек была добавлена в glibc в версии 2.4.

СООТВЕТСТВУЕТ top

stat(), **fstat**(), **lstat**(): SVr4, 4.3BSD, POSIX.1-2001, POSIX.1.2008.

fstatat(): POSIX.1-2008.

Согласно POSIX.1-2001, **lstat**() по символической ссылке должен возвращать действительную информацию только в поле st_size и тип файла поля st_mode структуры *stat*. POSIX.1-2008 ужесточает спецификацию, требуя, **чтобы lstat**() возвращал допустимую информацию во всех полях, кроме битов режима в *st_mode*.

Использование *st_blocks* и *st_blksize* поля могут быть менее переносимыми. (Они были введены в BSD. Интерпретация отличается между системами и, возможно, в одной системе, когда задействованы монтирования NFS.)

NOTES top

Поля меток времени

Старые ядра и старые стандарты не поддерживали наносекундные поля временных меток. Вместо этого было три поля временной метки— *st_atime*, *st_mtime*и *st_ctime*— типизированы как *time_t*, которые записывают временные метки с точностью до одной секунды.

Начиная с ядра 2.5.48, структура *stat* поддерживает наносекундное разрешение для трех полей метки времени файла. Наносекундные компоненты каждой метки времени доступны через имена формы *st_atim.tv_nsec*, если определены подходящие макросы тестирования функций. Наносекундные временные метки были стандартизированы в POSIX.1-2008, и, начиная с версии 2.12, glibc предоставляет имена наносекундных компонентов, если **_POSIX_C_SOURCE** определен со значением 200809L или выше, или **_XOPEN_SOURCE** определяется со значением 700 или больше. До glibc 2.19 включительно определения компонентов nanoseconds также определяются, если **_BSD_SOURCE** или **_SVID_SOURCE** определяется. Если ни один из вышеупомянутых макросов

не определен, то значения наносекунды отображаются с именами вида *st_atimensec*.

Библиотека C / различия в ядре

Со временем увеличение размера структуры *stat* привело к появлению трех последовательных версий **stat**(): *sys_stat*() (slot *__NR_oldstat*), *sys_newstat*() (slot *__NR_stat*) и *sys_stat64()* (слот *__NR_stat64*) на 32-разрядных платформах, таких как i386. Первые две версии уже присутствовали в Linux 1.0 (хотя и с разными именами); последняя была добавлена в Linux 2.4. Аналогичные замечания применимы к **fstat**() и **lstat**().

Ядро-внутренние версии структуры *stat*, рассматриваемые различными версиями, соответственно:

__old_kernel_stat
Оригинальная структура, с довольно узкими полями и без заполнения.

stat Больше *st_ino* поля и отступы добавлены в различные части структуры для дальнейшего расширения.

stat64 Еще большее поле st_ino, большие *поля st_uid* и *st_gid* для размещения Linux-2.4 расширения UID и GID до 32 бит, а также различные другие увеличенные поля и дальнейшее дополнение в структуре. (Различные байты заполнения в конечном итоге были использованы в Linux 2.6 с появлением 32-битных идентификаторов устройств и наносекундных компонентов для полей временных меток.)

Статистика glibc() функция-оболочка скрывает эти сведения от приложений, вызывая самую последнюю версию системного вызова , предоставленного ядром, и переупаковывая возвращенную информацию, если это требуется для старых двоичных файлов.

На современных 64-битных системах жизнь проще: есть один системный вызов stat() и ядро имеют дело со структурой stat, содержащей поля достаточного размера.

Базовый системный вызов, используемый glibc **fstatat**() функция-оболочка на самом деле называется **fstatat64**() или, в некоторых архитектурах, **newfstatat**() .

ПРИМЕРЫ сверху

Следующая программа вызывает **lstat**() и отображает выбранные поля в возвращаемом *stat* структура.

```
#включить <sys/types.h>
#включить <sys/stat.h>
#включить <stdint.h>
#включить <time.h>
#включить <stdio.h>
#включить <stdlib.h>
#включить <sys/sysmacros.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 (lstat(argv[1], &sb) == -1) {
perror ("lstat");
exit(EXIT_FAILURE);
}
```

```
 printf("Идентификатор содержащего устройства: [%jx,%jx]\n",
(uintmax_t) major(sb.st_dev),
(uintmax_t) minor(sb.st_dev));

 printf("Тип файла: ");

 switch (sb.st_mode & S_IFMT) {
case S_IFBLK: printf("block device\n"); break;
 case S_IFCHR: printf("символьное устройство\n"); break;
 case S_IFDIR: printf("каталог\n"); break;
 case S_IFIFO: printf("FIFO/pipe\n"); break;
 case S_IFLNK: printf("symlink\n"); break;
 case S_IFREG: printf("обычный файл\n"); break;
 case S_IFSOCK: printf("socket\n"); break;
 по умолчанию: printf("неизвестно?\n"); break;
 }

 printf("Номер I-узла: %ju\n", (uintmax_t) sb.st_ino);

 printf("Mode: %jo (octal)\n",
(uintmax_t) sb.st_mode);

 printf("Количество ссылок: %ju\n", (uintmax_t) sb.st_nlink);
 printf("Владение: UID=%ju GID=%ju\n",
(uintmax_t) sb.st_uid, (uintmax_t) sb.st_gid);

 printf("Предпочтительный размер блока ввода-вывода: %jd bytes\n",
(intmax_t) sb.st_blksize);
 printf("Размер файла: %jd bytes\n",
(intmax_t) sb.st_size);
 printf("Выделенные блоки: %jd\n",
(intmax_t) sb.st_blocks);

 printf("Последнее изменение статуса: %s", ctime(&sb.st_ctime));
```

```
 printf("Последний доступ к файлу: %s", ctime(&sb.st_atime));
 printf("Последняя модификация файла: %s", ctime(&sb.st_mtime));

 exit(EXIT_SUCCESS);
}
```

СМ. ТАКЖЕ top

ls(1), stat(1), доступ(2), chmod(2), chown(2), readlink(2), statx(2), utime(2), возможности(7), inode(7), символическая ссылка (7)

COLOPHON top

Эта страница является частью версии 5.13 проекта Linux *man-pages*.
Описание проекта, сведения об ошибках
и последнюю версию этой страницы можно найти по адресу
https://www.kernel.org/doc/man-pages /.

Linux 2021-08-27 STAT (2)

Страницы, которые ссылаются на эту страницу: bash(1), найти(1), git-update-index(1), pv(1), rsync(1), stat(1), strace(1), доступ(2), chmod(2), fallocate(2), fanotify_init(2), futimesat(2), getxattr(2), ioctl_ns(2), ссылка(2), listxattr(2), mkdir(2), mknod(2), mount(2), открыть (2), pivot_root(2), readlink(2), removexattr (2), setxattr(2), spu_create(2), statfs(2), statx(2), символическая ссылка (2), системные вызовы (2), truncate(2), umask(2), ustat(2), utime(2), utimensat(2), dirfd(3), euidaccess(3), fseek(3), ftok(3), fts(3), ftw(3), getfilecon(3), getseuserbyname(3), glob(3), isatty(3), isfdtype(3), makedev(3), mkfifo(3), readdir(3), readline(3), selabel_lookup_best_match(3), setfilecon(3), shm_open(3), ttyname(3), предохранитель (4), nfs(5), proc(5), selabel_file(5), sysfs(5), inode(7), inotify(7), пространства имен (7), path_resolution(7), pipe(7), shm_overview(7), сигнал-безопасность (7), spufs(7), символическая ссылка (7), system_data_types(7), время (7), user_namespaces(7), xattr(7), lsof(8), umount(8), xfs_db(8), xfs_io(8)