**int register\_blkdev(unsigned int***major***, const char \****name***);**

ARGUMENTS

*major*

запрашиваемый основной номер устройства

the requested major device number [1..255]. If *major*=0, try to allocate any unused major number.

попробуйте выделить какой-либо неиспользованный мажорный номер

*name*

the name of the new block device as a zero terminated string

имя нового блочного устройства в виде строки с нулевым символом в конце

DESCRIPTION

The *name* must be unique within the system.

The return value depends on the *major* input parameter. - if a major device number was requested in range [1..255] then the function returns zero on success, or a negative error code - if any unused major number was requested with *major*=0 parameter then the return value is the allocated major number in range [1..255] or a negative error code otherwise

406/5000

Имя должно быть уникальным в системе.  
Возвращаемое значение зависит от основного входного параметра. - если основной номер устройства был запрошен в диапазоне [1..255], то функция возвращает ноль при успешном завершении или отрицательный код ошибки - если какой-либо неиспользуемый основной номер был запрошен с параметром major = 0, тогда возвращаемое значение является назначенным основным число в диапазоне [1..255] или отрицательный код ошибки в противном случае

/dev/zero is an example of a "special file" — particularly, a "device node". Normally these get created by the distro installation process, but you can *totally* create them yourself if you want to.

Обычно они создаются в процессе установки дистрибутива, но вы можете полностью создать их самостоятельно, если хотите.

If you ask ls about /dev/zero:

# ls -l /dev/zero

crw-rw-rw- 1 root root 1, 5 Nov 5 09:34 /dev/zero

The "c" at the start tells you that this is a "character device"; the other type is "block device" (printed by ls as "b"). Very roughly, random-access devices like harddisks tend to be block devices, while sequential things like tape drives or your sound card tend to be character devices.

The "1, 5" part is the "major device number" and the "minor device number".

«С» в начале говорит вам, что это «символьное устройство»; другой тип - «блочное устройство» (печатается ls как «b»). Грубо говоря, устройства с произвольным доступом, такие как жесткие диски, как правило, являются блочными устройствами, в то время как последовательные устройства, такие как ленточные накопители или звуковая карта, обычно являются символьными устройствами.

Часть «1, 5» - это «основной номер устройства» и «вспомогательный номер устройства».

With this information, we can use the mknod command to make our very own device node:

Имея эту информацию, мы можем использовать команду mknod для создания нашего собственного узла устройства:

# mknod foobar c 1 5

This creates a new file named foobar, in the current folder, which does *exactly* the same thing as /dev/zero. (You can of course set different permissions on it if you want.) All this "file" really contains is the three items above — device type, major number, minor number. You can use ls to look up the codes for other devices and recreate those too. When you get bored, just use rm to remove the device nodes you just created.

Basically the major number tells the Linux kernel which device driver to talk to, and the minor number tells the device driver which device you're talking about. (E.g., you probably have one SATA controller, but maybe multiple harddisks plugged into it.)

Это создаст новый файл с именем foobar в текущей папке, который делает то же самое, что и / dev / zero. (Конечно, вы можете установить для него различные разрешения, если хотите.) Все, что действительно содержит этот «файл», это три элемента выше - тип устройства, основной номер, вспомогательный номер. Вы можете использовать ls для поиска кодов для других устройств и их воссоздания. Когда вам надоест, просто используйте rm для удаления только что созданных узлов устройства.

В основном, старший номер сообщает ядру Linux, с каким драйвером устройства следует обращаться, а младший номер указывает драйверу устройства, о каком устройстве вы говорите. (Например, у вас, вероятно, один контроллер SATA, но, возможно, к нему подключено несколько жестких дисков.)

If you want to *invent* new devices that do something new... well, you'll need to edit the source code for the Linux kernel and compile your own custom kernel. So let's not do that! :-) But you can add device files that duplicate the ones you've already got just fine. An automated system like udev is basically just watching for device events and calling mknod / rm for you automatically. Nothing more magic than that.

There are still *other* kinds of special files:

Если вы хотите изобретать новые устройства, которые делают что-то новое ... ну, вам нужно отредактировать исходный код для ядра Linux и скомпилировать собственное ядро. Так что давайте не будем этого делать! :-) Но вы можете добавить файлы устройств, которые дублируют те, которые вы уже получили, просто отлично. Автоматическая система, такая как udev, просто отслеживает события устройства и автоматически вызывает mknod / rm. Ничего более волшебного, чем это.

Есть еще другие виды специальных файлов:

* Linux considers a directory to be a special kind of file. (Usually you can't directly open a directory, but if you could, you'd find it's a normal file that contains data in a special format, and tells the kernel where to find all the files in that directory.)
* A symlink is a special file. (But a hard link isn't.) You can create symlinks using the ln -s command. (Look up the manpage for it.)
* There's also a thing called a "named pipe" or "FIFO" (first-in, first-out queue). You can create one with mkfifo. A FIFO is a magical file that can be opened by *two* programs at once — one reading, one writing. When this happens, it works like a normal shell pipe. But you can start each program separately...

A file that isn't "special" in any way is called a "regular file". You will occasionally see mention of this in Unix documentation. That's what it means; a file that isn't a device node or a symlink or whatever. Just a normal, every day file with no magical properties.

---------------------------------------------------------------------------------------

**SYNOPSIS**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

**#include <sys/sysmacros.h>**

**dev\_t makedev(unsigned int** *maj***, unsigned int** *min***);**

**unsigned int major(dev\_t** *dev***);**

**unsigned int minor(dev\_t** *dev***);**

**DESCRIPTION**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

A device ID consists of two parts: a major ID, identifying the class

of the device, and a minor ID, identifying a specific instance of a

device in that class. A device ID is represented using the type

*dev\_t*.

Given major and minor device IDs, **makedev**() combines these to produce

a device ID, returned as the function result. This device ID can be

given to [mknod(2)](http://man7.org/linux/man-pages/man2/mknod.2.html), for example.

The **major**() and **minor**() functions perform the converse task: given a

device ID, they return, respectively, the major and minor components.

These macros can be useful to, for example, decompose the device IDs

in the structure returned by [stat(2)](http://man7.org/linux/man-pages/man2/stat.2.html).

Идентификатор устройства состоит из двух частей: major ID, идентифицирующего класс устройства, и minor ID, идентифицирующего конкретный экземпляр устройства в этом классе. Идентификатор устройства представлен с использованием типа dev\_t.

        Учитывая major и minor идентификаторы устройств, функция makedev() объединяет их для получения идентификатора устройства, возвращаемого как результат функции. Этот идентификатор устройства можно присвоить, например, mknod (2).

Функции major () и minor () выполняют обратную задачу: при наличии идентификатора устройства они возвращают, соответственно, главный и вспомогательный компоненты.

        Эти макросы могут быть полезны, например, для декомпозиции идентификаторов устройств в структуре, возвращаемой stat (2).

**ATTRIBUTES**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

For an explanation of the terms used in this section, see

[attributes(7)](http://man7.org/linux/man-pages/man7/attributes.7.html).

┌────────────────────────────┬───────────────┬─────────┐

│**Interface** │ **Attribute** │ **Value** │

├────────────────────────────┼───────────────┼─────────┤

│**makedev**(), **major**(), **minor**() │ Thread safety │ MT-Safe │

└────────────────────────────┴───────────────┴─────────┘

**CONFORMING TO**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

The **makedev**(), **major**(), and **minor**() functions are not specified in

POSIX.1, but are present on many other systems.

**NOTES**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

These interfaces are defined as macros. Since glibc 2.3.3, they have

been aliases for three GNU-specific functions: **gnu\_dev\_makedev**(),

**gnu\_dev\_major**(), and **gnu\_dev\_minor**(). The latter names are exported,

but the traditional names are more portable.

The BSDs expose the definitions for these macros via *<sys/types.h>*.

Depending on the version, glibc also exposes definitions for these

macros from that header file if suitable feature test macros are

defined. However, this behavior was deprecated in glibc 2.25, and

since glibc 2.28, *<sys/types.h>* no longer provides these definitions.

Эти интерфейсы определены как макросы. Начиная с glibc 2.3.3 они были псевдонимами для трех специфичных для GNU функций: gnu\_dev\_makedev (), gnu\_dev\_major () и gnu\_dev\_minor (). Последние имена экспортируются, но традиционные имена более переносимы.

        BSD предоставляют определения этих макросов через <sys / types.h>. В зависимости от версии, glibc также предоставляет определения для этих макросов из этого заголовочного файла, если определены подходящие макросы тестирования возможностей. Однако в glibc 2.25 такое поведение устарело, и, поскольку glibc 2.28, <sys / types.h> больше не предоставляет эти определения.

**SEE ALSO**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

[mknod(2)](http://man7.org/linux/man-pages/man2/mknod.2.html), [stat(2)](http://man7.org/linux/man-pages/man2/stat.2.html)

**COLOPHON**[**top**](http://man7.org/linux/man-pages/man3/major.3.html#top_of_page)

This page is part of release 5.06 of the Linux *man-pages* project. A

description of the project, information about reporting bugs, and the

latest version of this page, can be found at

<https://www.kernel.org/doc/man-pages/>.

**The stat structure**

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; /\* File type and mode \*/

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; /\* Block size for filesystem I/O \*/

blkcnt\_t st\_blocks; /\* Number of 512B blocks allocated \*/

/\* Since Linux 2.6, the kernel supports nanosecond

precision for the following timestamp fields.

For the details before Linux 2.6, see NOTES. \*/

struct timespec st\_atim; /\* Time of last access \*/

struct timespec st\_mtim; /\* Time of last modification \*/

struct timespec st\_ctim; /\* Time of last status change \*/

#define st\_atime st\_atim.tv\_sec /\* Backward compatibility \*/

#define st\_mtime st\_mtim.tv\_sec

#define st\_ctime st\_ctim.tv\_sec

};

*Note*: the order of fields in the *stat* structure varies somewhat

across architectures. In addition, the definition above does not

show the padding bytes that may be present between some fields on

various architectures. Consult the glibc and kernel source code if

you need to know the details.

*Note*: for performance and simplicity reasons, different fields in the

*stat* structure may contain state information from different moments

during the execution of the system call. For example, if *st\_mode* or

*st\_uid* is changed by another process by calling [chmod(2)](http://man7.org/linux/man-pages/man2/chmod.2.html) or [chown(2)](http://man7.org/linux/man-pages/man2/chown.2.html),

**stat**() might return the old *st\_mode* together with the new *st\_uid*, or

the old *st\_uid* together with the new *st\_mode*.

The fields in the *stat* structure are as follows:

*st\_dev* This field describes the device on which this file resides.

(The [major(3)](http://man7.org/linux/man-pages/man3/major.3.html) and [minor(3)](http://man7.org/linux/man-pages/man3/minor.3.html) macros may be useful to decompose

the device ID in this field.)

*st\_ino* This field contains the file's inode number.

*st\_mode*

This field contains the file type and mode. See [inode(7)](http://man7.org/linux/man-pages/man7/inode.7.html) for

further information.

*st\_nlink*

This field contains the number of hard links to the file.

*st\_uid* This field contains the user ID of the owner of the file.

*st\_gid* This field contains the ID of the group owner of the file.

*st\_rdev*

This field describes the device that this file (inode) repre‐

sents.

*st\_size*

This field gives the size of the file (if it is a regular file

or a symbolic link) in bytes. The size of a symbolic link is

the length of the pathname it contains, without a terminating

null byte.

*st\_blksize*

This field gives the "preferred" block size for efficient

filesystem I/O.

*st\_blocks*

This field indicates the number of blocks allocated to the

file, in 512-byte units. (This may be smaller than

*st\_size*/512 when the file has holes.)

*st\_atime*

This is the time of the last access of file data.

*st\_mtime*

This is the time of last modification of file data.

*st\_ctime*

This is the file's last status change timestamp (time of last

change to the inode).

For further information on the above fields, see [inode(7)](http://man7.org/linux/man-pages/man7/inode.7.html).

Each file has an inode containing metadata about the file. An

application can retrieve this metadata using [stat(2)](http://man7.org/linux/man-pages/man2/stat.2.html) (or related

calls), which returns a *stat* structure, or [statx(2)](http://man7.org/linux/man-pages/man2/statx.2.html), which returns a

*statx* structure.

The following is a list of the information typically found in, or

associated with, the file inode, with the names of the corresponding

structure fields returned by [stat(2)](http://man7.org/linux/man-pages/man2/stat.2.html) and [statx(2)](http://man7.org/linux/man-pages/man2/statx.2.html):

Device where inode resides

Каждый файл имеет inode, содержащий метаданные о файле. Приложение может извлечь эти метаданные, используя stat (2) (или связанные вызовы), который возвращает структуру stat, или statx (2), который возвращает структуру statx.

        Ниже приведен список информации, обычно находящейся в файле inode или ассоциированной с ним, с именами соответствующих полей структуры, возвращаемых stat (2) и statx (2):

        Устройство, в котором находится индекс

*stat.st\_dev*; *statx.stx\_dev\_minor* and *statx.stx\_dev\_major*

Each inode (as well as the associated file) resides in a

filesystem that is hosted on a device. That device is

identified by the combination of its major ID (which

identifies the general class of device) and minor ID (which

identifies a specific instance in the general class).

Каждый inode (как и связанный с ним файл) находится в файловой системе, размещенной на устройстве. Это устройство идентифицируется комбинацией его major идентификатора (который идентифицирует общий класс устройства) и minor идентификатора (который идентифицирует конкретный экземпляр в общем классе).

Inode number

*stat.st\_ino*; *statx.stx\_ino*

Each file in a filesystem has a unique inode number. Inode

numbers are guaranteed to be unique only within a filesystem

(i.e., the same inode numbers may be used by different

filesystems, which is the reason that hard links may not cross

filesystem boundaries). This field contains the file's inode

number.

Каждый файл в файловой системе имеет уникальный номер inode. Номера inode гарантированно будут уникальными только внутри файловой системы (то есть, одни и те же inode могут использоваться разными файловыми системами, что является причиной того, что жесткие ссылки не могут пересекать границы файловой системы). Это поле содержит номер inode файла.

File type and mode

*stat.st\_mode*; *statx.stx\_mode*

See the discussion of file type and mode, below.

Link count

*stat.st\_nlink*; *statx.stx\_nlink*

This field contains the number of hard links to the file.

Additional links to an existing file are created using

[link(2)](http://man7.org/linux/man-pages/man2/link.2.html).

User ID

*st\_uid stat.st\_uid*; *statx.stx\_uid*

This field records the user ID of the owner of the file. For

newly created files, the file user ID is the effective user ID

of the creating process. The user ID of a file can be changed

using [chown(2)](http://man7.org/linux/man-pages/man2/chown.2.html).

Group ID

*stat.st\_gid*; *statx.stx\_gid*

The inode records the ID of the group owner of the file. For

newly created files, the file group ID is either the group ID

of the parent directory or the effective group ID of the

creating process, depending on whether or not the set-group-ID

bit is set on the parent directory (see below). The group ID

of a file can be changed using [chown(2)](http://man7.org/linux/man-pages/man2/chown.2.html).

Device represented by this inode

*stat.st\_rdev*; *statx.stx\_rdev\_minor* and *statx.stx\_rdev\_major*

If this file (inode) represents a device, then the inode

records the major and minor ID of that device.

Если этот файл (inode) представляет устройство, тогда inode записывает major и minor идентификатор этого устройства.

File size

*stat.st\_size*; *statx.stx\_size*

This field gives the size of the file (if it is a regular file

or a symbolic link) in bytes. The size of a symbolic link is

the length of the pathname it contains, without a terminating

null byte.

Preferred block size for I/O

*stat.st\_blksize*; *statx.stx\_blksize*

This field gives the "preferred" blocksize for efficient

filesystem I/O. (Writing to a file in smaller chunks may

cause an inefficient read-modify-rewrite.)

Number of blocks allocated to the file

*stat.st\_blocks*; *statx.stx\_size*

This 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 POSIX.1 standard notes that the unit for the *st\_blocks*

member of the *stat* structure is not defined by the standard.

On many implementations it is 512 bytes; on a few systems, a

different unit is used, such as 1024. Furthermore, the unit

may differ on a per-filesystem basis.

Last access timestamp (atime)

*stat.st\_atime*; *statx.stx\_atime*

This is the file's last access timestamp. It is changed by

file accesses, for example, by [execve(2)](http://man7.org/linux/man-pages/man2/execve.2.html), [mknod(2)](http://man7.org/linux/man-pages/man2/mknod.2.html), [pipe(2)](http://man7.org/linux/man-pages/man2/pipe.2.html),

[utime(2)](http://man7.org/linux/man-pages/man2/utime.2.html), and [read(2)](http://man7.org/linux/man-pages/man2/read.2.html) (of more than zero bytes). Other

interfaces, such as [mmap(2)](http://man7.org/linux/man-pages/man2/mmap.2.html), may or may not update the atime

timestamp

Some filesystem types allow mounting in such a way that file

and/or directory accesses do not cause an update of the atime

timestamp. (See *noatime*, *nodiratime*, and *relatime* in

[mount(8)](http://man7.org/linux/man-pages/man8/mount.8.html), and related information in [mount(2)](http://man7.org/linux/man-pages/man2/mount.2.html).) In addition,

the atime timestamp is not updated if a file is opened with

the **O\_NOATIME** flag; see [open(2)](http://man7.org/linux/man-pages/man2/open.2.html).

File creation (birth) timestamp (btime)

(not returned in the *stat* structure); *statx.stx\_btime*

The file's creation timestamp. This is set on file creation

and not changed subsequently.

The btime timestamp was not historically present on UNIX

systems and is not currently supported by most Linux

filesystems.

Last modification timestamp (mtime)

*stat.st\_mtime*; *statx.stx\_mtime*

This is the file's last modification timestamp. It is changed

by file modifications, for example, by [mknod(2)](http://man7.org/linux/man-pages/man2/mknod.2.html), [truncate(2)](http://man7.org/linux/man-pages/man2/truncate.2.html),

[utime(2)](http://man7.org/linux/man-pages/man2/utime.2.html), and [write(2)](http://man7.org/linux/man-pages/man2/write.2.html) (of more than zero bytes). Moreover,

the mtime timestamp of a directory is changed by the creation

or deletion of files in that directory. The mtime timestamp

is *not* changed for changes in owner, group, hard link count,

or mode.

Last status change timestamp (ctime)

*stat.st\_ctime*; *statx.stx\_ctime*

This is the file's last status change timestamp. It is

changed by writing or by setting inode information (i.e.,

owner, group, link count, mode, etc.).

The timestamp fields report time measured with a zero point at the

*Epoch*, 1970-01-02 00:00:00 +0000, UTC (see [time(7)](http://man7.org/linux/man-pages/man7/time.7.html)).

Nanosecond timestamps are supported on XFS, JFS, Btrfs, and ext4

(since Linux 2.6.23). Nanosecond timestamps are not supported in

ext2, ext3, and Reiserfs. In order to return timestamps with

nanosecond precision, the timestamp fields in the *stat* and *statx*

structures are defined as structures that include a nanosecond

component. See [stat(2)](http://man7.org/linux/man-pages/man2/stat.2.html) and [statx(2)](http://man7.org/linux/man-pages/man2/statx.2.html) for details. On filesystems

that do not support subsecond timestamps, the nanosecond fields in

the *stat* and *statx* structures are returned with the value 0.

**The file type and mode**

The *stat.st\_mode* field (for [statx(2)](http://man7.org/linux/man-pages/man2/statx.2.html), the *statx.stx\_mode* field)

contains the file type and mode.

POSIX refers to the *stat.st\_mode* bits corresponding to the mask

**S\_IFMT** (see below) as the *file type*, the 12 bits corresponding to the

mask 07777 as the *file mode bits* and the least significant 9 bits

(0777) as the *file permission bits*.

The following mask values are defined for the file type:

POSIX ссылается на биты stat.st\_mode, соответствующие маске S\_IFMT (см. Ниже), в качестве типа файла, 12 битов, соответствующих маске 07777, в качестве битов режима файла и наименьших 9 бит (0777) в качестве битов разрешения файла.

   Следующие значения маски определены для типа файла:

**S\_IFMT** 0170000 bit mask for the file type bit field

**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

Thus, to test for a regular file (for example), one could write:

stat(pathname, &sb);

if ((sb.st\_mode & S\_IFMT) == S\_IFREG) {

/\* Handle regular file \*/

}

Because tests of the above form are common, additional macros are

defined by POSIX to allow the test of the file type in *st\_mode* to be

written more concisely:

Поскольку тесты описанной выше формы являются общими, POSIX определяет дополнительные макросы, позволяющие писать тест типа файла в st\_mode более кратко:

**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.)

The preceding code snippet could thus be rewritten as:

Таким образом, предыдущий фрагмент кода можно переписать так:

stat(pathname, &sb);

if (S\_ISREG(sb.st\_mode)) {

/\* Handle regular file \*/

}

The definitions of most of the above file type test macros are pro‐

vided if any of the following feature test macros is defined:

**\_BSD\_SOURCE** (in glibc 2.19 and earlier), **\_SVID\_SOURCE** (in glibc 2.19

and earlier), or **\_DEFAULT\_SOURCE** (in glibc 2.20 and later). In addi‐

tion, definitions of all of the above macros except **S\_IFSOCK** and

**S\_ISSOCK**() are provided if **\_XOPEN\_SOURCE** is defined.

Определения большинства вышеупомянутых макросов тестирования типов файлов предоставляются, если определен любой из следующих макросов тестирования функций: \_BSD\_SOURCE (в glibc 2.19 и более ранних версиях), \_SVID\_SOURCE (в glibc 2.19 и более ранних версиях) или \_DEFAULT\_SOURCE (в glibc 2.20 и более поздних версиях) ). Кроме того, определения всех вышеупомянутых макросов, кроме S\_IFSOCK и S\_ISSOCK (), предоставляются, если определено \_XOPEN\_SOURCE.

The definition of **S\_IFSOCK** can also be exposed either by defining

**\_XOPEN\_SOURCE** with a value of 500 or greater or (since glibc 2.24) by

defining both **\_XOPEN\_SOURCE** and **\_XOPEN\_SOURCE\_EXTENDED**.

The definition of **S\_ISSOCK**() is exposed if any of the following fea‐

ture test macros is defined: **\_BSD\_SOURCE** (in glibc 2.19 and earlier),

**\_DEFAULT\_SOURCE** (in glibc 2.20 and later), **\_XOPEN\_SOURCE** with a value

of 500 or greater, **\_POSIX\_C\_SOURCE** with a value of 200112L or

greater, or (since glibc 2.24) by defining both **\_XOPEN\_SOURCE** and

**\_XOPEN\_SOURCE\_EXTENDED**.

The following mask values are defined for the file mode component of

the *st\_mode* field:

**S\_ISUID** 04000 set-user-ID bit (see [execve(2)](http://man7.org/linux/man-pages/man2/execve.2.html))

**S\_ISGID** 02000 set-group-ID bit (see below)

**S\_ISVTX** 01000 sticky bit (see below)

**S\_IRWXU** 00700 owner has read, write, and execute permission

**S\_IRUSR** 00400 owner has read permission

**S\_IWUSR** 00200 owner has write permission

**S\_IXUSR** 00100 owner has execute permission

**S\_IRWXG** 00070 group has read, write, and execute permission

**S\_IRGRP** 00040 group has read permission

**S\_IWGRP** 00020 group has write permission

**S\_IXGRP** 00010 group has execute permission

**S\_IRWXO** 00007 others (not in group) have read, write, and

execute permission

**S\_IROTH** 00004 others have read permission

**S\_IWOTH** 00002 others have write permission

**S\_IXOTH** 00001 others have execute permission

The set-group-ID bit (**S\_ISGID**) has several special uses. For a

directory, it indicates that BSD semantics are to be used for that

directory: files created there inherit their group ID from the direc‐

tory, not from the effective group ID of the creating process, and

directories created there will also get the **S\_ISGID** bit set. For an

executable file, the set-group-ID bit causes the effective group ID

of a process that executes the file to change as described in

[execve(2)](http://man7.org/linux/man-pages/man2/execve.2.html). 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.

## CONFORMING TO         [top](http://man7.org/linux/man-pages/man7/inode.7.html#top_of_page)

If you need to obtain the definition of the *blkcnt\_t* or *blksize\_t*

types from *<sys/stat.h>*, then define **\_XOPEN\_SOURCE** with the value 500

or greater (before including *any* header files).

POSIX.1-1990 did not describe the **S\_IFMT**, **S\_IFSOCK**, **S\_IFLNK**, **S\_IFREG**,

**S\_IFBLK**, **S\_IFDIR**, **S\_IFCHR**, **S\_IFIFO**, **S\_ISVTX** constants, but instead

specified the use of the macros **S\_ISDIR**(), and so on. The **S\_IF\***

constants are present in POSIX.1-2001 and later.

The **S\_ISLNK**() and **S\_ISSOCK**() macros were 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**.

Если вам нужно получить определение типов blkcnt\_t или blksize\_t из <sys / stat.h>, то определите \_XOPEN\_SOURCE со значением 500 или больше (перед включением любых заголовочных файлов).

        POSIX.1-1990 не описывал константы S\_IFMT, S\_IFSOCK, S\_IFLNK, S\_IFREG, S\_IFBLK, S\_IFDIR, S\_IFCHR, S\_IFIFO, S\_ISVTX, но вместо этого определял использование макросов S\_ISDIR () и т. Д. Константы S\_IF \* присутствуют в POSIX.1-2001 и более поздних версиях.

        Макросы S\_ISLNK () и S\_ISSOCK () отсутствовали в POSIX.1-1996, но оба присутствуют в POSIX.1-2001; первый из SVID 4, последний из SUSv2.

        UNIX V7 (и более поздние системы) имели S\_IREAD, S\_IWRITE, S\_IEXEC, где POSIX предписывает синонимы S\_IRUSR, S\_IWUSR, S\_IXUSR.

## NOTES         [top](http://man7.org/linux/man-pages/man7/inode.7.html#top_of_page)

For pseudofiles that are autogenerated by the kernel, the file size

(*stat.st\_size*; *statx.stx\_size*) reported by the kernel is not

accurate. For example, the value 0 is returned for many files under

the */proc* directory, while various files under */sys* report a size of

4096 bytes, even though the file content is smaller. For such files,

one should simply try to read as many bytes as possible (and append

'\0' to the returned buffer if it is to be interpreted as a string).

**fstatat()**

The **fstatat**() system call is a more general interface for accessing

file information which can still provide exactly the behavior of each

of **stat**(), **lstat**(), and **fstat**().

If the pathname given in *pathname* is relative, then it is interpreted

relative to the directory referred to by the file descriptor *dirfd*

(rather than relative to the current working directory of the calling

process, as is done by **stat**() and **lstat**() for a relative pathname).

If *pathname* is relative and *dirfd* is the special value **AT\_FDCWD**, then

*pathname* is interpreted relative to the current working directory of

the calling process (like **stat**() and **lstat**()).

If *pathname* is absolute, then *dirfd* is ignored.

*flags* can either be 0, or include one or more of the following flags

ORed:

struct [**request\_queue**](https://elixir.bootlin.com/linux/latest/ident/request_queue);

struct [**block\_device**](https://elixir.bootlin.com/linux/latest/ident/block_device) {

[**dev\_t**](https://elixir.bootlin.com/linux/latest/ident/dev_t) bd\_dev; */\* not a kdev\_t - it's a search key \*/*

int bd\_openers;

struct [**inode**](https://elixir.bootlin.com/linux/latest/ident/inode) \* bd\_inode; */\* will die \*/*

struct [**super\_block**](https://elixir.bootlin.com/linux/latest/ident/super_block) \* bd\_super;

struct [**mutex**](https://elixir.bootlin.com/linux/latest/ident/mutex) bd\_mutex; */\* open/close mutex \*/*

void \* bd\_claiming;

void \* [**bd\_holder**](https://elixir.bootlin.com/linux/latest/ident/bd_holder);

int bd\_holders;

[**bool**](https://elixir.bootlin.com/linux/latest/ident/bool) bd\_write\_holder;

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_SYSFS

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) bd\_holder\_disks;

#endif

struct [**block\_device**](https://elixir.bootlin.com/linux/latest/ident/block_device) \* bd\_contains;

unsigned bd\_block\_size;

[**u8**](https://elixir.bootlin.com/linux/latest/ident/u8) bd\_partno;

struct [**hd\_struct**](https://elixir.bootlin.com/linux/latest/ident/hd_struct) \* bd\_part;

*/\* number of times partitions within this device have been opened. \*/*

unsigned bd\_part\_count;

int bd\_invalidated;

struct [**gendisk**](https://elixir.bootlin.com/linux/latest/ident/gendisk) \* bd\_disk;

struct [**request\_queue**](https://elixir.bootlin.com/linux/latest/ident/request_queue) \* bd\_queue;

struct [**backing\_dev\_info**](https://elixir.bootlin.com/linux/latest/ident/backing_dev_info) \*bd\_bdi;

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) [**bd\_list**](https://elixir.bootlin.com/linux/latest/ident/bd_list);

*/\**

*\* Private data. You must have bd\_claim'ed the block\_device*

*\* to use this. NOTE: bd\_claim allows an owner to claim*

*\* the same device multiple times, the owner must take special*

*\* care to not mess up bd\_private for that case.*

*\*/*

unsigned long bd\_private;

*/\* The counter of freeze processes \*/*

int bd\_fsfreeze\_count;

*/\* Mutex for freeze \*/*

struct [**mutex**](https://elixir.bootlin.com/linux/latest/ident/mutex) bd\_fsfreeze\_mutex;

} [**\_\_randomize\_layout**](https://elixir.bootlin.com/linux/latest/ident/__randomize_layout);

*/\**

*\* Keep mostly read-only and often accessed (especially for*

*\* the RCU path lookup and 'stat' data) fields at the beginning*

*\* of the 'struct inode'*

*\*/*

struct [**inode**](https://elixir.bootlin.com/linux/latest/ident/inode) {

[**umode\_t**](https://elixir.bootlin.com/linux/latest/ident/umode_t) i\_mode;

unsigned short i\_opflags;

[**kuid\_t**](https://elixir.bootlin.com/linux/latest/ident/kuid_t) i\_uid;

[**kgid\_t**](https://elixir.bootlin.com/linux/latest/ident/kgid_t) i\_gid;

unsigned int i\_flags;

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_FS\_POSIX\_ACL

struct [**posix\_acl**](https://elixir.bootlin.com/linux/latest/ident/posix_acl) \*i\_acl;

struct [**posix\_acl**](https://elixir.bootlin.com/linux/latest/ident/posix_acl) \*i\_default\_acl;

#endif

const struct [**inode\_operations**](https://elixir.bootlin.com/linux/latest/ident/inode_operations) \*i\_op;

struct [**super\_block**](https://elixir.bootlin.com/linux/latest/ident/super_block) \*i\_sb;

struct [**address\_space**](https://elixir.bootlin.com/linux/latest/ident/address_space) \*i\_mapping;

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_SECURITY

void \*i\_security;

#endif

*/\* Stat data, not accessed from path walking \*/*

unsigned long i\_ino;

*/\**

*\* Filesystems may only read i\_nlink directly. They shall use the*

*\* following functions for modification:*

*\**

*\* (set|clear|inc|drop)\_nlink*

*\* inode\_(inc|dec)\_link\_count*

*\*/*

union {

const unsigned int i\_nlink;

unsigned int \_\_i\_nlink;

};

[**dev\_t**](https://elixir.bootlin.com/linux/latest/ident/dev_t) i\_rdev;

[**loff\_t**](https://elixir.bootlin.com/linux/latest/ident/loff_t) i\_size;

struct [**timespec64**](https://elixir.bootlin.com/linux/latest/ident/timespec64) i\_atime;

struct [**timespec64**](https://elixir.bootlin.com/linux/latest/ident/timespec64) i\_mtime;

struct [**timespec64**](https://elixir.bootlin.com/linux/latest/ident/timespec64) i\_ctime;

[**spinlock\_t**](https://elixir.bootlin.com/linux/latest/ident/spinlock_t) i\_lock; */\* i\_blocks, i\_bytes, maybe i\_size \*/*

unsigned short i\_bytes;

[**u8**](https://elixir.bootlin.com/linux/latest/ident/u8) i\_blkbits;

[**u8**](https://elixir.bootlin.com/linux/latest/ident/u8) i\_write\_hint;

[**blkcnt\_t**](https://elixir.bootlin.com/linux/latest/ident/blkcnt_t) i\_blocks;

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) [**\_\_NEED\_I\_SIZE\_ORDERED**](https://elixir.bootlin.com/linux/latest/ident/__NEED_I_SIZE_ORDERED)

[**seqcount\_t**](https://elixir.bootlin.com/linux/latest/ident/seqcount_t) i\_size\_seqcount;

#endif

*/\* Misc \*/*

unsigned long i\_state;

struct [**rw\_semaphore**](https://elixir.bootlin.com/linux/latest/ident/rw_semaphore) i\_rwsem;

unsigned long dirtied\_when; */\* jiffies of first dirtying \*/*

unsigned long dirtied\_time\_when;

struct [**hlist\_node**](https://elixir.bootlin.com/linux/latest/ident/hlist_node) i\_hash;

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) i\_io\_list; */\* backing dev IO list \*/*

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_CGROUP\_WRITEBACK

struct [**bdi\_writeback**](https://elixir.bootlin.com/linux/latest/ident/bdi_writeback) \*i\_wb; */\* the associated cgroup wb \*/*

*/\* foreign inode detection, see wbc\_detach\_inode() \*/*

int i\_wb\_frn\_winner;

[**u16**](https://elixir.bootlin.com/linux/latest/ident/u16) i\_wb\_frn\_avg\_time;

[**u16**](https://elixir.bootlin.com/linux/latest/ident/u16) i\_wb\_frn\_history;

#endif

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) i\_lru; */\* inode LRU list \*/*

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) i\_sb\_list;

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) i\_wb\_list; */\* backing dev writeback list \*/*

union {

struct [**hlist\_head**](https://elixir.bootlin.com/linux/latest/ident/hlist_head) i\_dentry;

struct [**rcu\_head**](https://elixir.bootlin.com/linux/latest/ident/rcu_head) i\_rcu;

};

[**atomic64\_t**](https://elixir.bootlin.com/linux/latest/ident/atomic64_t) i\_version;

[**atomic64\_t**](https://elixir.bootlin.com/linux/latest/ident/atomic64_t) i\_sequence; */\* see futex \*/*

[**atomic\_t**](https://elixir.bootlin.com/linux/latest/ident/atomic_t) i\_count;

[**atomic\_t**](https://elixir.bootlin.com/linux/latest/ident/atomic_t) i\_dio\_count;

[**atomic\_t**](https://elixir.bootlin.com/linux/latest/ident/atomic_t) i\_writecount;

#if [**defined**](https://elixir.bootlin.com/linux/latest/ident/defined)(CONFIG\_IMA) || [**defined**](https://elixir.bootlin.com/linux/latest/ident/defined)(CONFIG\_FILE\_LOCKING)

[**atomic\_t**](https://elixir.bootlin.com/linux/latest/ident/atomic_t) i\_readcount; */\* struct files open RO \*/*

#endif

union {

const struct [**file\_operations**](https://elixir.bootlin.com/linux/latest/ident/file_operations) \*i\_fop; */\* former ->i\_op->default\_file\_ops \*/*

void (\***[free\_inode](https://elixir.bootlin.com/linux/latest/ident/free_inode)**)(struct [**inode**](https://elixir.bootlin.com/linux/latest/ident/inode) \*);

};

struct [**file\_lock\_context**](https://elixir.bootlin.com/linux/latest/ident/file_lock_context) \*i\_flctx;

struct [**address\_space**](https://elixir.bootlin.com/linux/latest/ident/address_space) [**i\_data**](https://elixir.bootlin.com/linux/latest/ident/i_data);

struct [**list\_head**](https://elixir.bootlin.com/linux/latest/ident/list_head) i\_devices;

union {

struct [**pipe\_inode\_info**](https://elixir.bootlin.com/linux/latest/ident/pipe_inode_info) \*i\_pipe;

struct [**block\_device**](https://elixir.bootlin.com/linux/latest/ident/block_device) \*i\_bdev;

struct [**cdev**](https://elixir.bootlin.com/linux/latest/ident/cdev) \*i\_cdev;

char \*i\_link;

unsigned i\_dir\_seq;

};

[**\_\_u32**](https://elixir.bootlin.com/linux/latest/ident/__u32) i\_generation;

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_FSNOTIFY

[**\_\_u32**](https://elixir.bootlin.com/linux/latest/ident/__u32) i\_fsnotify\_mask; */\* all events this inode cares about \*/*

struct [**fsnotify\_mark\_connector**](https://elixir.bootlin.com/linux/latest/ident/fsnotify_mark_connector) [**\_\_rcu**](https://elixir.bootlin.com/linux/latest/ident/__rcu) \*i\_fsnotify\_marks;

#endif

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_FS\_ENCRYPTION

struct [**fscrypt\_info**](https://elixir.bootlin.com/linux/latest/ident/fscrypt_info) \*i\_crypt\_info;

#endif

#[**ifdef**](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_FS\_VERITY

struct [**fsverity\_info**](https://elixir.bootlin.com/linux/latest/ident/fsverity_info) \*i\_verity\_info;

#endif

void \*i\_private; */\* fs or device private pointer \*/*

} [**\_\_randomize\_layout**](https://elixir.bootlin.com/linux/latest/ident/__randomize_layout);

/\*\*

\* struct device - The basic device structure

\* @parent: The device's "parent" device, the device to which it is attached.

\* In most cases, a parent device is some sort of bus or host

\* controller. If parent is NULL, the device, is a top-level device,

\* which is not usually what you want.

\* @p: Holds the private data of the driver core portions of the device.

\* See the comment of the struct device\_private for detail.

\* @kobj: A top-level, abstract class from which other classes are derived. Абстрактный класс верхнего уровня, из которого получены другие классы.

\* @init\_name: Initial name of the device. Начальное название устройства.

\* @type: The type of device.

\* This identifies the device type and carries type-specific

\* information. Тип устройства. Это идентифицирует тип устройства и несет специфическую для типа информацию

\* @mutex: Mutex to synchronize calls to its driver.

\* @lockdep\_mutex: An optional debug lock that a subsystem can use as a

\* peer lock to gain localized lockdep coverage of the device\_lock. Необязательная блокировка отладки, которую подсистема может использовать в качестве одноранговой блокировки для получения локализованного покрытия lockdep для device\_lock.

\* @bus: Type of bus device is on.

\* @driver: Which driver has allocated this

\* @platform\_data: Platform data specific to the device.

\* Example: For devices on custom boards, as typical of embedded

\* and SOC based hardware, Linux often uses platform\_data to point

\* to board-specific structures describing devices and how they

\* are wired. That can include what ports are available, chip

\* variants, which GPIO pins act in what additional roles, and so

\* on. This shrinks the "Board Support Packages" (BSPs) and

\* minimizes board-specific #ifdefs in drivers.

Данные платформы, специфичные для устройства. Пример. Для устройств на пользовательских платах, типичных для встроенного оборудования и оборудования на базе SOC, Linux часто использует platform\_data, чтобы указывать на специфические для платы структуры, описывающие устройства и их проводную связь. Это может включать в себя то, какие порты доступны, варианты микросхем, какие контакты GPIO играют в каких дополнительных ролях и так далее. Это сокращает «Пакеты поддержки платы» (BSP) и минимизирует специфичные для платы #ifdefs в драйверах.

\* @driver\_data: Private pointer for driver specific info.

\* @links: Links to suppliers and consumers of this device. Ссылки на поставщиков и потребителей этого устройства.

\* @power: For device power management.

\* See Documentation/driver-api/pm/devices.rst for details.

\* @pm\_domain: Provide callbacks that are executed during system suspend,

\* hibernation, system resume and during runtime PM transitions

\* along with subsystem-level and driver-level callbacks.

Обеспечивает обратные вызовы, которые выполняются во время приостановки системы, гибернации, возобновления системы и во время переходов PM во время выполнения наряду с обратными вызовами уровня подсистемы и уровня драйвера.

\* @pins: For device pin management.

\* See Documentation/driver-api/pinctl.rst for details.

\* @msi\_list: Hosts MSI descriptors Хосты MSI дескрипторы

\* @msi\_domain: The generic MSI domain this device is using.

\* @numa\_node: NUMA node this device is close to.

\* @dma\_ops: DMA mapping operations for this device.

\* @dma\_mask: Dma mask (if dma'ble device).

\* @coherent\_dma\_mask: Like dma\_mask, but for alloc\_coherent mapping as not all

\* hardware supports 64-bit addresses for consistent allocations

\* such descriptors.

Подобно dma\_mask, но для сопоставления alloc\_coherent, поскольку не все аппаратные средства поддерживают 64-битные адреса для согласованного распределения таких дескрипторов.

\* @bus\_dma\_limit: Limit of an upstream bridge or bus which imposes a smaller

\* DMA limit than the device itself supports.

Предел верхнего моста или шины, который накладывает меньший предел DMA, чем само устройство поддерживает.

\* @dma\_pfn\_offset: offset of DMA memory range relatively of RAM

\* @dma\_parms: A low level driver may set these to teach IOMMU code about

\* segment limitations.

\* @dma\_pools: Dma pools (if dma'ble device).

\* @dma\_mem: Internal for coherent mem override.

\* @cma\_area: Contiguous memory area for dma allocations

\* @archdata: For arch-specific additions.

\* @of\_node: Associated device tree node.

\* @fwnode: Associated device node supplied by platform firmware.

\* @devt: For creating the sysfs "dev".

\* @id: device instance

\* @devres\_lock: Spinlock to protect the resource of the device.

\* @devres\_head: The resources list of the device.

\* @knode\_class: The node used to add the device to the class list.

\* @class: The class of the device.

\* @groups: Optional attribute groups.

\* @release: Callback to free the device after all references have

\* gone away. This should be set by the allocator of the

\* device (i.e. the bus driver that discovered the device).

\* @iommu\_group: IOMMU group the device belongs to.

\* @iommu\_fwspec: IOMMU-specific properties supplied by firmware.

\* @iommu\_param: Per device generic IOMMU runtime data

\*

\* @offline\_disabled: If set, the device is permanently online.

\* @offline: Set after successful invocation of bus type's .offline().

\* @of\_node\_reused: Set if the device-tree node is shared with an ancestor

\* device.

\* @state\_synced: The hardware state of this device has been synced to match

\* the software state of this device by calling the driver/bus

\* sync\_state() callback.

Аппаратное состояние этого устройства было синхронизировано в соответствии с программным состоянием этого устройства путем вызова обратного вызова driver / bus sync\_state ().

\* @dma\_coherent: this particular device is dma coherent, even if the

\* architecture supports non-coherent devices.

это конкретное устройство является когерентным по dma, даже если архитектура поддерживает некогерентные устройства.

\*

\* At the lowest level, every device in a Linux system is represented by an

\* instance of struct device. The device structure contains the information

\* that the device model core needs to model the system. Most subsystems,

\* however, track additional information about the devices they host. As a

\* result, it is rare for devices to be represented by bare device structures;

\* instead, that structure, like kobject structures, is usually embedded within

\* a higher-level representation of the device.

\*/

На самом низком уровне каждое устройство в системе Linux представлено экземпляром struct device. Структура устройства содержит информацию, которая необходима ядру модели устройства для моделирования системы. Большинство подсистем, однако, отслеживают дополнительную информацию об устройствах, которые они размещают. В результате устройства редко представляются голыми структурами устройства; вместо этого эта структура, как и структуры kobject, обычно встроена в представление устройства более высокого уровня.

At the lowest level, every device in a Linux system is represented by an instance of struct device. The device structure contains the information that the device model core needs to model the system. Most subsystems, however, track additional information about the devices they host. As a result, it is rare for devices to be represented by bare device structures; instead, that structure, like kobject structures, is usually embedded within a higher-level representation of the device.

На самом низком уровне каждое устройство в системе Linux представлено экземпляром struct device. Структура устройства содержит информацию, которая необходима ядру модели устройства для моделирования системы. Большинство подсистем, однако, отслеживают дополнительную информацию об устройствах, которые они размещают. В результате устройства редко представляются голыми структурами устройства; вместо этого эта структура, как и структуры kobject, обычно встроена в представление устройства более высокого уровня.

struct [device](https://elixir.bootlin.com/linux/latest/ident/device) {

struct [kobject](https://elixir.bootlin.com/linux/latest/ident/kobject) kobj;

struct [device](https://elixir.bootlin.com/linux/latest/ident/device) \*[parent](https://elixir.bootlin.com/linux/latest/ident/parent);

struct [device\_private](https://elixir.bootlin.com/linux/latest/ident/device_private) \*p;

const char \*init\_name; /\* initial name of the device \*/

const struct [device\_type](https://elixir.bootlin.com/linux/latest/ident/device_type) \*type; /\* The type of device. This identifies the device type and carries type-specific information.\*/

struct [bus\_type](https://elixir.bootlin.com/linux/latest/ident/bus_type) \*[bus](https://elixir.bootlin.com/linux/latest/ident/bus); /\* type of bus device is on \*/

struct [device\_driver](https://elixir.bootlin.com/linux/latest/ident/device_driver) \*driver; /\* which driver has allocated this device \*/

void \*[platform\_data](https://elixir.bootlin.com/linux/latest/ident/platform_data); /\* Platform specific data, device

core doesn't touch it \*/

void \*[driver\_data](https://elixir.bootlin.com/linux/latest/ident/driver_data); /\* Driver data, set and get with

dev\_set\_drvdata/dev\_get\_drvdata \*/

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_PROVE\_LOCKING

struct [mutex](https://elixir.bootlin.com/linux/latest/ident/mutex) lockdep\_mutex;

#endif

struct [mutex](https://elixir.bootlin.com/linux/latest/ident/mutex) [mutex](https://elixir.bootlin.com/linux/latest/ident/mutex); /\* mutex to synchronize calls to

\* its driver.

\*/

struct [dev\_links\_info](https://elixir.bootlin.com/linux/latest/ident/dev_links_info) links;

struct [dev\_pm\_info](https://elixir.bootlin.com/linux/latest/ident/dev_pm_info) [power](https://elixir.bootlin.com/linux/latest/ident/power);

struct [dev\_pm\_domain](https://elixir.bootlin.com/linux/latest/ident/dev_pm_domain) \*[pm\_domain](https://elixir.bootlin.com/linux/latest/ident/pm_domain);

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_GENERIC\_MSI\_IRQ\_DOMAIN

struct [irq\_domain](https://elixir.bootlin.com/linux/latest/ident/irq_domain) \*msi\_domain;

#endif

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_PINCTRL

struct [dev\_pin\_info](https://elixir.bootlin.com/linux/latest/ident/dev_pin_info) \*[pins](https://elixir.bootlin.com/linux/latest/ident/pins);

#endif

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_GENERIC\_MSI\_IRQ

struct [list\_head](https://elixir.bootlin.com/linux/latest/ident/list_head) msi\_list;

#endif

const struct [dma\_map\_ops](https://elixir.bootlin.com/linux/latest/ident/dma_map_ops) \*[dma\_ops](https://elixir.bootlin.com/linux/latest/ident/dma_ops);

[u64](https://elixir.bootlin.com/linux/latest/ident/u64) \*[dma\_mask](https://elixir.bootlin.com/linux/latest/ident/dma_mask); /\* dma mask (if dma'able device) \*/

[u64](https://elixir.bootlin.com/linux/latest/ident/u64) coherent\_dma\_mask;/\* Like dma\_mask, but for

alloc\_coherent mappings as

not all hardware supports

64 bit addresses for consistent

allocations such descriptors. \*/

[u64](https://elixir.bootlin.com/linux/latest/ident/u64) bus\_dma\_limit; /\* upstream dma constraint \*/

unsigned long [dma\_pfn\_offset](https://elixir.bootlin.com/linux/latest/ident/dma_pfn_offset);

struct [device\_dma\_parameters](https://elixir.bootlin.com/linux/latest/ident/device_dma_parameters) \*dma\_parms;

struct [list\_head](https://elixir.bootlin.com/linux/latest/ident/list_head) dma\_pools; /\* dma pools (if dma'ble) \*/

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_DMA\_DECLARE\_COHERENT

struct [dma\_coherent\_mem](https://elixir.bootlin.com/linux/latest/ident/dma_coherent_mem) \*dma\_mem; /\* internal for coherent mem

override \*/

#endif

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_DMA\_CMA

struct [cma](https://elixir.bootlin.com/linux/latest/ident/cma) \*cma\_area; /\* contiguous memory area for dma

allocations \*/

#endif

/\* arch specific additions \*/

struct [dev\_archdata](https://elixir.bootlin.com/linux/latest/ident/dev_archdata) archdata;

struct [device\_node](https://elixir.bootlin.com/linux/latest/ident/device_node) \*[of\_node](https://elixir.bootlin.com/linux/latest/ident/of_node); /\* associated device tree node \*/

struct [fwnode\_handle](https://elixir.bootlin.com/linux/latest/ident/fwnode_handle) \*fwnode; /\* firmware device node \*/

#[ifdef](https://elixir.bootlin.com/linux/latest/ident/ifdef) CONFIG\_NUMA

int [numa\_node](https://elixir.bootlin.com/linux/latest/ident/numa_node); /\* NUMA node this device is close to \*/

#endif

[dev\_t](https://elixir.bootlin.com/linux/latest/ident/dev_t) devt; /\* dev\_t, creates the sysfs "dev" \*/

[u32](https://elixir.bootlin.com/linux/latest/ident/u32) id; /\* device instance \*/

[spinlock\_t](https://elixir.bootlin.com/linux/latest/ident/spinlock_t) devres\_lock;

struct [list\_head](https://elixir.bootlin.com/linux/latest/ident/list_head) devres\_head;

struct [class](https://elixir.bootlin.com/linux/latest/ident/class) \*[class](https://elixir.bootlin.com/linux/latest/ident/class);

const struct [attribute\_group](https://elixir.bootlin.com/linux/latest/ident/attribute_group) \*\*[groups](https://elixir.bootlin.com/linux/latest/ident/groups); /\* optional groups \*/

void (\*[release](https://elixir.bootlin.com/linux/latest/ident/release))(struct [device](https://elixir.bootlin.com/linux/latest/ident/device) \*dev);

struct [iommu\_group](https://elixir.bootlin.com/linux/latest/ident/iommu_group) \*[iommu\_group](https://elixir.bootlin.com/linux/latest/ident/iommu_group);

struct [iommu\_fwspec](https://elixir.bootlin.com/linux/latest/ident/iommu_fwspec) \*[iommu\_fwspec](https://elixir.bootlin.com/linux/latest/ident/iommu_fwspec);

struct [iommu\_param](https://elixir.bootlin.com/linux/latest/ident/iommu_param) \*[iommu\_param](https://elixir.bootlin.com/linux/latest/ident/iommu_param);

[bool](https://elixir.bootlin.com/linux/latest/ident/bool) offline\_disabled:1;

[bool](https://elixir.bootlin.com/linux/latest/ident/bool) [offline](https://elixir.bootlin.com/linux/latest/ident/offline):1;

[bool](https://elixir.bootlin.com/linux/latest/ident/bool) of\_node\_reused:1;

[bool](https://elixir.bootlin.com/linux/latest/ident/bool) state\_synced:1;

#if [defined](https://elixir.bootlin.com/linux/latest/ident/defined)(CONFIG\_ARCH\_HAS\_SYNC\_DMA\_FOR\_DEVICE) || \

[defined](https://elixir.bootlin.com/linux/latest/ident/defined)(CONFIG\_ARCH\_HAS\_SYNC\_DMA\_FOR\_CPU) || \

[defined](https://elixir.bootlin.com/linux/latest/ident/defined)(CONFIG\_ARCH\_HAS\_SYNC\_DMA\_FOR\_CPU\_ALL)

[bool](https://elixir.bootlin.com/linux/latest/ident/bool) dma\_coherent:1;

#endif

};

struct device\_driver

The basic device driver structure

Definition

struct device\_driver {

const char \* name;

struct bus\_type \* bus;

struct module \* owner;

const char \* mod\_name;

bool suppress\_bind\_attrs;

enum probe\_type probe\_type;

const struct of\_device\_id \* of\_match\_table;

const struct acpi\_device\_id \* acpi\_match\_table;

int (\* probe) (struct device \*dev);

int (\* remove) (struct device \*dev);

void (\* shutdown) (struct device \*dev);

int (\* suspend) (struct device \*dev, pm\_message\_t state);

int (\* resume) (struct device \*dev);

const struct attribute\_group \*\* groups;

const struct dev\_pm\_ops \* pm;

struct driver\_private \* p;

};

Members

**name** - Name of the device driver.

**bus** - The bus which the device of this driver belongs to.

**owner** - The module owner.

mod\_name - Used for built-in modules.

suppress\_bind\_attrs - Disables bind/unbind via sysfs.

probe\_type - Type of the probe (synchronous or asynchronous) to use.

of\_match\_table - The open firmware table.

acpi\_match\_table - The ACPI match table.

**probe** - Called to query the existence of a specific device, whether this driver can work with it, and bind the driver to a specific device.

**remove** - Called when the device is removed from the system to unbind a device from this driver.

**shutdown** - Called at shut-down time to quiesce the device.

**suspend** - Called to put the device to sleep mode. Usually to a low power state.

**resume** - Called to bring a device from sleep mode.

**groups** - Default attributes that get created by the driver core automatically.

**pm** - Power management operations of the device which matched this driver.

**p** - Driver core’s private data, no one other than the driver core can touch this.

**Description**

The device driver-model tracks all of the drivers known to the system. The main reason for this tracking is to enable the driver core to match up drivers with new devices. Once drivers are known objects within the system, however, a number of other things become possible. Device drivers can export information and configuration variables that are independent of any specific device.

Модель драйвера устройства отслеживает все драйверы, известные системе. Основная причина такого отслеживания заключается в том, чтобы ядро драйвера соответствовало драйверам с новыми устройствами. Однако, как только драйверы становятся известными объектами в системе, становится возможным ряд других вещей. Драйверы устройств могут экспортировать информацию и переменные конфигурации, которые не зависят от конкретного устройства.