**Linux内核编程接口函数**

**草根老师：http://blog.chinaunix.net/uid/26833883/abstract/1.html**

转载请注明出处: <http://blog.csdn.net/drivelinux/article/details/8656280>

**字符设备相关函数**

**1.alloc\_chrdev\_region()**

功能: 自动分配一个主设备号及基于此主设备号的若干个连续的指定数量的次设备号。

函数原型如下:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* alloc\_chrdev\_region() - register a range of char device numbers
3. \* @dev: output parameter for first assigned number
4. \* @baseminor: first of the requested range of minor numbers
5. \* @count: the number of minor numbers required
6. \* @name: the name of the associated device or driver
7. \*
8. \* Allocates a range of char device numbers.  The major number will be
9. \* chosen dynamically, and returned (along with the first minor number)
10. \* in @dev.  Returns zero or a negative error code.
11. \*/
12. **int** alloc\_chrdev\_region(dev\_t \*dev, unsigned baseminor, unsigned count,**const** **char** \*name)

**2.register\_chrdev\_region()**

功能:注册连续的若干个设备号，用这个函数要自己指定主设备号。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* register\_chrdev\_region() - register a range of device numbers
3. \* @from: the first in the desired range of device numbers; must include
4. \*        the major number.
5. \* @count: the number of consecutive device numbers required
6. \* @name: the name of the device or driver.
7. \*
8. \* Return value is zero on success, a negative error code on failure.
9. \*/
10. **int** register\_chrdev\_region(dev\_t from, unsigned count, **const** **char** \*name)

注： register\_chrdev() 函数也用于分配设备号，但是用于2.6版本之前的内核中，在之后的版本中不再使用。

**3.unregister\_chrdev\_region()**

功能:注销由alloc\_chrdev\_region()或者register\_chrdev\_region()注册的设备号，一般在驱动模块的退出函数中要执行此操作，将注册的设备号还给系统。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* unregister\_chrdev\_region() - return a range of device numbers
3. \* @from: the first in the range of numbers to unregister
4. \* @count: the number of device numbers to unregister
5. \*
6. \* This function will unregister a range of @count device numbers,
7. \* starting with @from.  The caller should normally be the one who
8. \* allocated those numbers in the first place...
9. \*/
10. **void** unregister\_chrdev\_region(dev\_t from, unsigned count)

**4.cdev\_alloc()**

功能:分配一个表示字符设备的cdev结构体。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* cdev\_alloc() - allocate a cdev structure
3. \*
4. \* Allocates and returns a cdev structure, or NULL on failure.
5. \*/
6. **struct** cdev \*cdev\_alloc(**void**)

**5.cdev\_init()**

功能:初始化由cdev\_alloc()分配的表示字符设备的cdev结构体，并通过向该函数传入的struct file\_operations \*fops参数来指定该字符设备的文件操作函数。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* cdev\_init() - initialize a cdev structure
3. \* @cdev: the structure to initialize
4. \* @fops: the file\_operations for this device
5. \*
6. \* Initializes @cdev, remembering @fops, making it ready to add to the
7. \* system with cdev\_add().
8. \*/
9. **void** cdev\_init(**struct** cdev \*cdev, **const** **struct** file\_operations \*fops)

**6.cdev\_add()**

功能:添加一个(由cdev\_alloc()分配并通过cdev\_init()初始化的)字符设备到系统中。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* cdev\_add() - add a char device to the system
3. \* @p: the cdev structure for the device
4. \* @dev: the first device number for which this device is responsible
5. \* @count: the number of consecutive minor numbers corresponding to this
6. \*         device
7. \*
8. \* cdev\_add() adds the device represented by @p to the system, making it
9. \* live immediately.  A negative error code is returned on failure.
10. \*/
11. **int** cdev\_add(**struct** cdev \*p, dev\_t dev, unsigned count)

**7.cdev\_del()**

功能:执行和cdev\_add()相反的操作，将一个指定的字符设备从系统中删除。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* cdev\_del() - remove a cdev from the system
3. \* @p: the cdev structure to be removed
4. \*
5. \* cdev\_del() removes @p from the system, possibly freeing the structure
6. \* itself.
7. \*/
8. **void** cdev\_del(**struct** cdev \*p)

以下为一个简单的字符设备初始化函数，函数中列出了向系统中添加一个字符设备的简单步骤:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. **struct** cdev \*myDev\_dev;
2. /\*主设备和从设备号变量\*/
3. **static** **int** myDev\_major = 0;
4. **static** **int** myDev\_minor = 0;
6. **static** **int** \_\_init myDev\_init(**void**)
7. {
8. **int** err = -1;
9. dev\_t dev = 0;
11. /\*1.动态分配主设备号和从设备号,第三个参数表示分配的此设备号的数量\*/
12. err = alloc\_chrdev\_region(&dev, 0, TOTAL\_DEVICE\_COUNT, "myDev");
13. **if**(err < 0) {
14. printk(KERN\_ALERT"Failed to alloc char dev region.\n");
15. **return** err;
16. }
18. myDev\_major = MAJOR(dev);
19. myDev\_minor = MINOR(dev);
21. /\*2.分配cdev字符设备结构体变量\*/
22. myDev\_dev = cdev\_alloc();
23. **if**(!myDev\_dev) {
24. err = -ENOMEM;
25. printk(KERN\_ALERT"Failed to alloc myDev\_dev.\n");
26. **return** err;
27. }
28. /\*3.初始化cdev字符设备结构体\*/
29. cdev\_init(myDev\_dev, &myDev\_fops);
31. /\*4.添加字符设备到系统中\*/
32. err = cdev\_add(myDev\_dev,dev, TOTAL\_DEVICE\_COUNT);
33. **if**(err) {
34. **return** err;
35. }
36. **return** err;
37. }

**8.register\_chrdev()**

这个函数也是实现注册字符设备的功能，但是相对于上面提到的alloc\_chrdev\_region()和register\_chrdev\_region()而言，是比较老的形式。并且register\_chrdev()一次能够注册统一主设备号下的256个不同的次设备。并且，在2.6之后的内核版本中不再使用这一函数。

**自动创建设备节点相关函数**

**1.class\_create()**

功能:创建一个struct class结构体，作为device\_create()函数的参数。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. #define class\_create(owner, name)       \
2. ({                      \
3. **static** **struct** lock\_class\_key \_\_key; \
4. \_\_class\_create(owner, name, &\_\_key);    \
5. })

**2.device\_create()**

功能:该函数创建一个设备并将其注册到sysfs中，同时在系统的sys/class和sys/device目录下会生成相应的类和设备入口。并且，该函数还会出发用户空间udev的动作，udev会根据sysfs下的class在/dev目录下创建设备节点，这也为自动创建设备节点提供了一种途径。通过device\_create()函数，我们就可以不用通过mknod命令手动的创建设备节点了。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* device\_create - creates a device and registers it with sysfs
3. \* @class: pointer to the struct class that this device should be registered to
4. \* @parent: pointer to the parent struct device of this new device, if any
5. \* @devt: the dev\_t for the char device to be added
6. \* @drvdata: the data to be added to the device for callbacks
7. \* @fmt: string for the device's name
8. \*
9. \* This function can be used by char device classes.  A struct device
10. \* will be created in sysfs, registered to the specified class.
11. \*
12. \* A "dev" file will be created, showing the dev\_t for the device, if
13. \* the dev\_t is not 0,0.
14. \* If a pointer to a parent struct device is passed in, the newly created
15. \* struct device will be a child of that device in sysfs.
16. \* The pointer to the struct device will be returned from the call.
17. \* Any further sysfs files that might be required can be created using this
18. \* pointer.
19. \*
20. \* Returns &struct device pointer on success, or ERR\_PTR() on error.
21. \*
22. \* Note: the struct class passed to this function must have previously
23. \* been created with a call to class\_create().
24. \*/
25. **struct** device \*device\_create(**struct** **class** \***class**, **struct** device \*parent,
26. dev\_t devt, **void** \*drvdata, **const** **char** \*fmt, ...)

**3.device\_create\_file()**

功能:该函数用于为指定设备在sysfs下建立属性文件，如：  
device\_create\_file(dev, &dev\_attr\_val);

其中的dev\_attr\_val可以通过DEVICE\_ATTR宏进行定义，也可以直接通过struct device\_attribute xxx=\_\_ATTR(\_name,\_mode,\_show,\_store)进行定义，区别就是用DEVICE\_ATTR宏进行定义时得到的变量名都是以dev\_attr\_为前缀的，而通过struct device\_attribute xxx=\_\_ATTR(\_name,\_mode,\_show,\_store)进行定义时，变量名可以任意指定。

函数原型:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* device\_create\_file - create sysfs attribute file for device.
3. \* @dev: device.
4. \* @attr: device attribute descriptor.
5. \*/
7. **int** device\_create\_file(**struct** device \*dev,**const** **struct** device\_attribute \*attr)

10. #define DEVICE\_ATTR(\_name, \_mode, \_show, \_store) \
11. **struct** device\_attribute dev\_attr\_##\_name = \_\_ATTR(\_name, \_mode, \_show, \_store)

如下代码可以实现自动创建设备节点:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. **static** **struct** **class**\* myDev\_class = NULL;
2. **struct** device\* devc= NULL;
4. /\*5.创建一个struct class结构体\*/
5. myDev\_class = class\_create(THIS\_MODULE,"myDevClass");
6. **if**(IS\_ERR(myDev\_class)) {
7. err = PTR\_ERR(myDev\_class);
8. printk(KERN\_ALERT"Failed to create  class.\n");
9. **return** err;
10. }
11. /\*6.在sysf中生成设备文件并在/dev下自动生成设备节点\*/
12. devc = device\_create(myDev\_class,NULL,dev,NULL,"myDev");
13. **if**(IS\_ERR(devc)) {
14. err = PTR\_ERR(devc);
15. printk(KERN\_ALERT"Failed to create device.");
16. **return** err;
17. }

以下贴出以上分两部分介绍的字符设备的初始化代码:

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. **struct** cdev \*myDev\_dev;
2. /\*主设备号和从设备号变量\*/
3. **static** **int** myDev\_major = 0;
4. **static** **int** myDev\_minor = 0;

7. **static** **int** \_\_init myDev\_init(**void**)
8. {
9. **int** err = -1;
10. dev\_t dev = 0;
11. **struct** device \*devc = NULL;
13. /\*1.动态分配主设备号和从设备号,第三个参数表示分配的此设备号的数量\*/
14. err = alloc\_chrdev\_region(&dev, 0, TOTAL\_DEVICE\_COUNT, "myDev");
15. **if**(err < 0) {
16. printk(KERN\_ALERT"Failed to alloc char dev region.\n");
17. **return** err;
18. }
20. myDev\_major = MAJOR(dev);
21. myDev\_minor = MINOR(dev);
23. /\*2.分配cdev字符设备结构体变量\*/
24. myDev\_dev = cdev\_alloc();
25. **if**(!myDev\_dev) {
26. err = -ENOMEM;
27. printk(KERN\_ALERT"Failed to alloc myDev\_dev.\n");
28. **return** err;
29. }
30. /\*3.初始化cdev字符设备结构体\*/
31. cdev\_init(myDev\_dev, &myDev\_fops);
33. /\*4.添加字符设备到系统中\*/
34. err = cdev\_add(myDev\_dev,dev, TOTAL\_DEVICE\_COUNT);
35. **if**(err) {
36. **return** err;
37. }
39. /\*5.创建一个struct class结构体\*/
40. myDev\_class = class\_create(THIS\_MODULE,"myDevClass");
41. **if**(IS\_ERR(myDev\_class)) {
42. err = PTR\_ERR(myDev\_class);
43. printk(KERN\_ALERT"Failed to create  class.\n");
44. **return** err;
45. }
46. /\*6.在sysf中生成设备文件并在/dev下自动生成设备节点\*/
47. devc = device\_create(myDev\_class,NULL,dev,NULL,"myDev");
48. **if**(IS\_ERR(devc)) {
49. err = PTR\_ERR(devc);
50. printk(KERN\_ALERT"Failed to create device.");
51. **return** err;
52. }
53. **return** err;
54. }

**通用函数**

**1.kmalloc()和kzalloc()的异同**

在内核include/linux/slab.h文件中有如下定义：

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* kzalloc - allocate memory. The memory is set to zero.
3. \* @size: how many bytes of memory are required.
4. \* @flags: the type of memory to allocate (see kmalloc).
5. \*/
6. **static** **inline** **void** \*kzalloc(**size\_t** size, gfp\_t flags)
7. {
8. **return** kmalloc(size, flags | \_\_GFP\_ZERO);
9. }

从注释以及函数原型均能看出kzalloc()函数的作用，即申请一块内存并且该内存会在函数调用过程中被清零。

**2.module\_platform\_driver()**

该函数实际是一个宏，它在include/linux/platform\_device.h中定义如下：

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\* module\_platform\_driver() - Helper macro for drivers that don't do
2. \* anything special in module init/exit.  This eliminates a lot of
3. \* boilerplate.  Each module may only use this macro once, and
4. \* calling it replaces module\_init() and module\_exit()
5. \*/
6. #define module\_platform\_driver(\_\_platform\_driver) \
7. module\_driver(\_\_platform\_driver, platform\_driver\_register, \
8. platform\_driver\_unregister)

其中的module\_driver在/include/linux/device.h中定义，如下：

**[cpp]** [view plain](http://blog.csdn.net/drivelinux/article/details/8656280) [copy](http://blog.csdn.net/drivelinux/article/details/8656280)

[在CODE上查看代码片](https://code.csdn.net/snippets/193358)

1. /\*\*
2. \* module\_driver() - Helper macro for drivers that don't do anything
3. \* special in module init/exit. This eliminates a lot of boilerplate.
4. \* Each module may only use this macro once, and calling it replaces
5. \* module\_init() and module\_exit().
6. \*
7. \* @\_\_driver: driver name
8. \* @\_\_register: register function for this driver type
9. \* @\_\_unregister: unregister function for this driver type
10. \* @...: Additional arguments to be passed to \_\_register and \_\_unregister.
11. \*
12. \* Use this macro to construct bus specific macros for registering
13. \* drivers, and do not use it on its own.
14. \*/
15. #define module\_driver(\_\_driver, \_\_register, \_\_unregister, ...) \
16. **static** **int** \_\_init \_\_driver##\_init(**void**) \
17. { \
18. **return** \_\_register(&(\_\_driver) , ##\_\_VA\_ARGS\_\_); \
19. } \
20. module\_init(\_\_driver##\_init); \
21. **static** **void** \_\_exit \_\_driver##\_exit(**void**) \
22. { \
23. \_\_unregister(&(\_\_driver) , ##\_\_VA\_ARGS\_\_); \
24. } \
25. module\_exit(\_\_driver##\_exit);

由上述定义可知，module\_platform\_driver()宏的作用就是定义指定名称的平台设备驱动注册函数和平台设备驱动注销函数，并且在函数体内分别通过platform\_driver\_register()函数和platform\_driver\_unregister()函数注册和注销该平台设备驱动。

**3.read\_cpuid\_id()**

读取CPU ID，定义于arch/arm/include/asm/cputype.h文件中。

**4.container\_of**