所有char device driver都被注册在如下hash table中管理

in fs/char_dev.c

```
static struct char_device_struct {
        struct char_device_struct *next;
        unsigned int major;
        unsigned int baseminor;
        int minorct;
        char name[64];
        struct cdev *cdev;              /* will die */
} *chrdevs[CHRDEV_MAJOR_HASH_SIZE];
```

```
#define CHRDEV_MAJOR_HASH_SIZE  255
```

chrdevs[CHRDEV_MAJOR_HASH_SIZE]是struct char_device_struct的指针数组。

其中dev_t(设备号)的major部分（高12位）用作进入该hash table的hash key.

```
static inline int major_to_index(unsigned major)
{
        return major % CHRDEV_MAJOR_HASH_SIZE;
}
```

即主设备号对255求余的结果为该table的index。

相同hash value的char_device_struct 通过next指针链接起来。

```
static struct char_device_struct {
        struct char_device_struct *next;
        unsigned int major;
        unsigned int baseminor;
        int minorct;
        char name[64];
        struct cdev *cdev;              /* will die */
};
```

baseminor, 该device driver管理的device的minior number是从baseminor开始的，一般应该是从0开始。

minorct应该是有该device driver管理着多少个device。

char device driver能管理的device的minior number必须是连续的，而不能是离散的。

即[baseminor, baseminor + minorct)

比如baseminor = 0，minorct =4

则device minior number is 0, 1, 2, 3

```
/*
 * Register a single major with a specified minor range.
 *
 * If major == 0 this functions will dynamically allocate a major and return
 * its number.
 *
 * If major > 0 this function will attempt to reserve the passed range of
 * minors and will return zero on success.
 *
 * Returns a -ve errno on failure.
 */
static struct char_device_struct *
__register_chrdev_region(unsigned int major, unsigned int baseminor,
                         int minorct, const char *name)
{
        struct char_device_struct *cd, **cp;
        int ret = 0;
        int i;

        cd = kzalloc(sizeof(struct char_device_struct), GFP_KERNEL);
        if (cd == NULL)
                return ERR_PTR(-ENOMEM);

        mutex_lock(&chrdevs_lock);

        /* temporary */
        if (major == 0) {                                                    ①
                for (i = ARRAY_SIZE(chrdevs)-1; i > 0; i--) {        ②
                        if (chrdevs[i] == NULL)
                                break;
                }

                if (i == 0) {
                        ret = -EBUSY;
                        goto out;
                }
                major = i;
                ret = major;
        }

        cd->major = major;
        cd->baseminor = baseminor;
        cd->minorct = minorct;
        strlcpy(cd->name, name, sizeof(cd->name));

        i = major_to_index(major);

        for (cp = &chrdevs[i]; *cp; cp = &(*cp)->next)            ③
                if ((*cp)->major > major ||
                    ((*cp)->major == major &&
                     (((*cp)->baseminor >= baseminor) ||
                      ((*cp)->baseminor + (*cp)->minorct > baseminor))))
                        break;
```

```
54.
55.          /* Check for overlapping minor ranges.  */
56.          if (*cp && (*cp)->major == major) {                ④
57.                  int old_min = (*cp)->baseminor;
58.                  int old_max = (*cp)->baseminor + (*cp)->minorct - 1;
59.                  int new_min = baseminor;
60.                  int new_max = baseminor + minorct - 1;
61.
62.                  /* New driver overlaps from the left.  */
63.                  if (new_max >= old_min && new_max <= old_max) {④
64.                          ret = -EBUSY;
65.                          goto out;
66.                  }
67.
68.                  /* New driver overlaps from the right.  */
69.                  if (new_min <= old_max && new_min >= old_min) {
70.                          ret = -EBUSY;
71.                          goto out;
72.                  }
73.          }
74.
75.          cd->next = *cp;                          ⑤
76.          *cp = cd;
77.          mutex_unlock(&chrdevs_lock);
78.          return cd;
79.  out:
80.          mutex_unlock(&chrdevs_lock);
81.          kfree(cd);
82.          return ERR_PTR(ret);
83.  }
```

①

If major == 0 this functions will dynamically allocate a major

②

从hash table的尾部开始搜索，所以返回的major是最大的空闲entry的index。即major的动态分配是从大往小分配的。

③

查找新的char_device_struct node要插入的地方。

相同hash value(也就是major在hash table中冲突)的char_device_struct node通过next field链接在一起，并且是按mojor的大从小刀到大排序的。

排序原则是：

1. major的大小，小的在前
2. major相同，则baseminor的大小，同样小的在前

```
(((*cp)->baseminor >= baseminor) ||
 ((*cp)->baseminor + (*cp)->minorct > baseminor))))
```

判断是否有overlap

④

如果两者的major相同，并且minor space有overlap，则报错

⑤

把new node插入lined-list。

---

struct char_device_struct主要是管理dev_t (设备号)用的。