in drivers/uio/uio.c

```
struct uio_listener {
        struct uio_device *dev;
        s32 event_count;
};
```

uio_listener干什么用的？

.dev field可以理解,用于把uio device file handle与uio_device之间建立关联。由file handle可以得到该file handle对应的uio_device

.event_count做何用？

---

uio_listener instance在uio_open()中生成

```
1.  static int uio_open(struct inode *inode, struct file *filep)
2.  {
3.          struct uio_device *idev;
4.          struct uio_listener *listener;
5.          int ret = 0;
6.
7.          mutex_lock(&minor_lock);
8.          idev = idr_find(&uio_idr, iminor(inode));
9.          mutex_unlock(&minor_lock);
10.         if (!idev) {
11.                 ret = -ENODEV;
12.                 goto out;
13.         }
14.
15.         if (!try_module_get(idev->owner)) {
16.                 ret = -ENODEV;
17.                 goto out;
18.         }
19.
20.         listener = kmalloc(sizeof(*listener), GFP_KERNEL);
21.         if (!listener) {
22.                 ret = -ENOMEM;
23.                 goto err_alloc_listener;
24.         }
25.
26.         listener->dev = idev;
27.         listener->event_count = atomic_read(&idev->event);        ①
28.         filep->private_data = listener;
29.
30.         if (idev->info->open) {
31.                 ret = idev->info->open(idev->info, inode);
32.                 if (ret)
33.                         goto err_infoopen;
34.         }
35.         return 0;
36.
37. err_infoopen:
38.         kfree(listener);
39.
40. err_alloc_listener:
41.         module_put(idev->owner);
42.
43. out:
44.         return ret;
45. }
```

①

uio_listener的.event_count记录的是当open uio device file时的event count(当时的该uio device的interrupt count)

uio_listener instance在uio_release()中被free

```
1.  static int uio_release(struct inode *inode, struct file *filep)
2.  {
3.          int ret = 0;
4.          struct uio_listener *listener = filep->private_data;
5.          struct uio_device *idev = listener->dev;
6.
7.          if (idev->info->release)
8.                  ret = idev->info->release(idev->info, inode);
9.
10.         module_put(idev->owner);
11.         kfree(listener);
12.         return ret;
13. }
```

在uio_read() / uio_poll()中又对listener->event_count的使用

```
1.  static ssize_t uio_read(struct file *filep, char __user *buf,
2.                         size_t count, loff_t *ppos)
3.  {
4.          struct uio_listener *listener = filep->private_data;
5.          struct uio_device *idev = listener->dev;
6.          DECLARE_WAITQUEUE(wait, current);
7.          ssize_t retval;
8.          s32 event_count;
9.
10.         if (!idev->info->irq)
11.                 return -EIO;
12.
13.         if (count != sizeof(s32))
14.                 return -EINVAL;
15.
16.         add_wait_queue(&idev->wait, &wait);
17.
18.         do {
19.                 set_current_state(TASK_INTERRUPTIBLE);
20.
21.                 event_count = atomic_read(&idev->event);
22. if (event_count != listener->event_count) {        ①
23.                         if (copy_to_user(buf, &event_count, count))
24.                                 retval = -EFAULT;
25.                         else {
26.                                 listener->event_count = event_count;          ②
27.                                 retval = count;
28.                         }
29.                         break;
30.                 }
31.
32.                 if (filep->f_flags & O_NONBLOCK) {
33.                         retval = -EAGAIN;
34.                         break;
35.                 }
36.
37.                 if (signal_pending(current)) {
38.                         retval = -ERESTARTSYS;
39.                         break;
40.                 }
41.                 schedule();
42.         } while (1);
43.
44.         __set_current_state(TASK_RUNNING);
45.         remove_wait_queue(&idev->wait, &wait);
46.
47.         return retval;
48. }
```

```
1.  static unsigned int uio_poll(struct file *filep, poll_table *wait)
2.  {
3.          struct uio_listener *listener = filep->private_data;
4.          struct uio_device *idev = listener->dev;
5.
6.          if (!idev->info->irq)
7.                  return -EIO;
8.
9.          poll_wait(filep, &idev->wait, wait);
10. if (listener->event_count != atomic_read(&idev->event))     ①
11.                 return POLLIN | POLLRDNORM;
12.         return 0;
13. }
```

①

listener->event_count != atomic_read(&idev->event)

不相等是由于在open uio device以后，有新的interrupt产生了。

listener->event_count记录的是正当open uio device时的interrupt count,

而idev->event是该uio device的实时的interrupt count.

②

为了下次read()能知道有新的interrupt产生，需要更新listener->event_count为uio device的最新interrupt count。