作者：董昊 （要转载的同学帮忙把名字和博客链接http://donghao.org/uii/带上，多谢了！）

epoll独有的EPOLLET

EPOLLET是epoll系统调用独有的flag，ET就是Edge Trigger（边缘触发）的意思，具体含义和应用大家可google之。有了EPOLLET，重复的事件就不会总是出来打扰程序的判断，故而常被使用。那EPOLLET的原理是什么呢？
上篇我们讲到epoll把fd都挂上一个回调函数，当fd对应的设备有消息时，就把fd放入rdllist链表，这样epoll_wait只要检查这个rdllist链表就可以知道哪些fd有事件了。我们看看ep_poll的最后几行代码：
[fs/eventpoll.c->ep_poll()]

```
 1524
 1525     /*
 1526      * Try to transfer events to user space. In case we get 0 events and
 1527      * there's still timeout left over, we go trying again in search of
 1528      * more luck.
 1529      */
 1530     if (!res && eavail &&
 1531         !(res = ep_events_transfer(ep, events, maxevents)) && jtimeout)
 1532         goto retry;
 1533
 1534     return res;
 1535 }
```

把rdllist里的fd拷到用户空间，这个任务是ep_events_transfer做的：

[fs/eventpoll.c->ep_events_transfer()]

```
 1439 static int ep_events_transfer(struct eventpoll *ep,
 1440                               struct epoll_event __user *events, int maxevents)
 1441 {
 1442     int eventcnt = 0;
 1443     struct list_head txlist;
 1444
 1445     INIT_LIST_HEAD(&txlist);
 1446
 1447     /*
 1448      * We need to lock this because we could be hit by
 1449      * eventpoll_release_file() and epoll_ctl(EPOLL_CTL_DEL).
 1450      */
 1451     down_read(&ep->sem);
 1452
 1453     /* Collect/extract ready items */
 1454     if (ep_collect_ready_items(ep, &txlist, maxevents) > 0) {
 1455         /* Build result set in userspace */
 1456         eventcnt = ep_send_events(ep, &txlist, events);
 1457
 1458         /* Reinject ready items into the ready list */
 1459         ep_reinject_items(ep, &txlist);
 1460     }
 1461
 1462     up_read(&ep->sem);
 1463
 1464     return eventcnt;
 1465 }
```

代码很少，其中ep_collect_ready_items把rdllist里的fd挪到txlist里（挪完后rdllist就空了），接着ep_send_events把txlist里的fd拷给用户空间，然后ep_reinject_items把一部分fd从txlist里“返还”给rdllist以便下次还能从rdllist里发现它。
其中ep_send_events的实现：

[fs/eventpoll.c->ep_send_events()]

```
1337 static int ep_send_events(struct eventpoll *ep, struct list_head *txlist,
1338                struct epoll_event __user *events)
1339 {
1340     int eventcnt = 0;
1341     unsigned int revents;
1342     struct list_head *lnk;
1343     struct epitem *epi;
1344
1345     /*
1346      * We can loop without lock because this is a task private list.
1347      * The test done during the collection loop will guarantee us that
1348      * another task will not try to collect this file. Also, items
1349      * cannot vanish during the loop because we are holding "sem".
1350      */
1351     list_for_each(lnk, txlist) {
1352         epi = list_entry(lnk, struct epitem, txlink);
1353
1354         /*
1355          * Get the ready file event set. We can safely use the file
1356          * because we are holding the "sem" in read and this will
1357          * guarantee that both the file and the item will not vanish.
1358          */
1359         revents = epi->ffd.file->f_op->poll(epi->ffd.file, NULL);
1360
1361         /*
1362          * Set the return event set for the current file descriptor.
1363          * Note that only the task task was successfully able to link
1364          * the item to its "txlist" will write this field.
1365          */
1366         epi->revents = revents & epi->event.events;
1367
1368         if (epi->revents) {
1369             if (__put_user(epi->revents,
1370                        &events[eventcnt].events) ||
1371                 __put_user(epi->event.data,
1372                        &events[eventcnt].data))
1373                 return -EFAULT;
1374             if (epi->event.events & EPOLLONESHOT)
1375                 epi->event.events &= EP_PRIVATE_BITS;
1376             eventcnt++;
1377         }
1378     }
1379     return eventcnt;
1380 }
```

这个拷贝实现其实没什么可看的，但是请注意1359行，这个poll很狡猾，它把第二个参数置为NULL来调用。我们先看一下设备驱动通常是怎么实现poll的：

```
static unsigned int scull_p_poll(struct file *filp, poll_table *wait)
{
	struct scull_pipe *dev = filp->private_data;
	unsigned int mask = 0;

	/*
	* The buffer is circular; it is considered full
	* if "wp" is right behind "rp" and empty if the
	* two are equal.
	*/
	down(&dev->sem);
	poll_wait(filp, &dev->inq,  wait);
	poll_wait(filp, &dev->outq, wait);
	if (dev->rp != dev->wp)
		mask |= POLLIN | POLLRDNORM;  /* readable */
	if (spacefree(dev))
		mask |= POLLOUT | POLLWRNORM;  /* writable */
	up(&dev->sem);
	return mask;
}
```

上面这段代码摘自《linux设备驱动程序（第三版）》，绝对经典，设备先要把current（当前进程）挂在inq和outq两个队列上（这个“挂”操作是wait回调函数指针做的），然后等设备来唤醒，唤醒后就能通过mask拿到事件掩码了（注意那个mask参数，它就是负责拿事件掩码的）。那如果wait为NULL，poll_wait会做些什么呢？

```
[include/linux/poll.h->poll_wait]
  25 static inline void poll_wait(struct file * filp, wait_queue_head_t * wait_address,
poll_table *p)
  26 {
  27     if (p && wait_address)
  28         p->qproc(filp, wait_address, p);
  29 }
```

喏，看见了，如果poll_table为空，什么也不做。我们倒回ep_send_events，那句标红的poll，实际上就是“我不想休眠，我只想拿到事件掩码”的意思。然后再把拿到的事件掩码拷给用户空间。
ep_send_events完成后，就轮到ep_reinject_items了：

```
[fs/eventpoll.c->ep_reinject_items]
 1389 static void ep_reinject_items(struct eventpoll *ep, struct list_head *txlist)
 1390 {
 1391     int ricnt = 0, pwake = 0;
 1392     unsigned long flags;
 1393     struct epitem *epi;
 1394
 1395     write_lock_irqsave(&ep->lock, flags);
 1396
 1397     while (!list_empty(txlist)) {
 1398         epi = list_entry(txlist->next, struct epitem, txlink);
 1399
```

```
1400        /* Unlink the current item from the transfer list */
1401        EP_LIST_DEL(&epi->txlink);
1402
1403        /*
1404         * If the item is no more linked to the interest set, we don't
1405         * have to push it inside the ready list because the following
1406         * ep_release_epitem() is going to drop it. Also, if the current
1407         * item is set to have an Edge Triggered behaviour, we don't have
1408         * to push it back either.
1409         */
1410        if (EP_RB_LINKED(&epi->rbn) && !(epi->event.events & EPOLLET) &&
1411            (epi->revents & epi->event.events) && !EP_IS_LINKED(&epi->rdllink)) {
1412            list_add_tail(&epi->rdllink, &ep->rdllist);
1413            ricnt++;
1414        }
1415    }
1416
1417    if (ricnt) {
1418        /*
1419         * Wake up ( if active ) both the eventpoll wait list and the ->poll()
1420         * wait list.
1421         */
1422        if (waitqueue_active(&ep->wq))
1423            wake_up(&ep->wq);
1424        if (waitqueue_active(&ep->poll_wait))
1425            pwake++;
1426    }
1427
1428    write_unlock_irqrestore(&ep->lock, flags);
1429
1430    /* We have to call this outside the lock */
1431    if (pwake)
1432        ep_poll_safewake(&psw, &ep->poll_wait);
1433 }
```

ep_reinject_items把txlist里的一部分fd又放回rdllist，那么，是把哪一部分fd放回去呢？看上面1410行的那个判断——是哪些"没有标上EPOLLET"（标红代码）且"事件被关注"（标蓝代码）的fd被重新放回了rdllist。那么下次epoll_wait当然会又把rdllist里的fd拿来拷给用户了。
举个例子。假设一个socket，只是connect，还没有收发数据，那么它的poll事件掩码总是有POLLOUT的（参见上面的驱动示例），每次调用epoll_wait总是返回POLLOUT事件（比较烦），因为它的fd就总是被放回rdllist；假如此时有人往这个socket里写了一大堆数据，造成socket塞住（不可写了），那么1411行里标蓝色的判断就不成立了（没有POLLOUT了），fd不会放回rdllist，epoll_wait将不会再返回用户POLLOUT事件。现在我们给这个socket加上EPOLLET，然后connect，没有收发数据，此时，1410行标红的判断又不成立了，所以epoll_wait只会返回一次POLLOUT通知给用户（因为此fd不会再回到rdllist了），接下来的epoll_wait都不会有任何事件通知了。