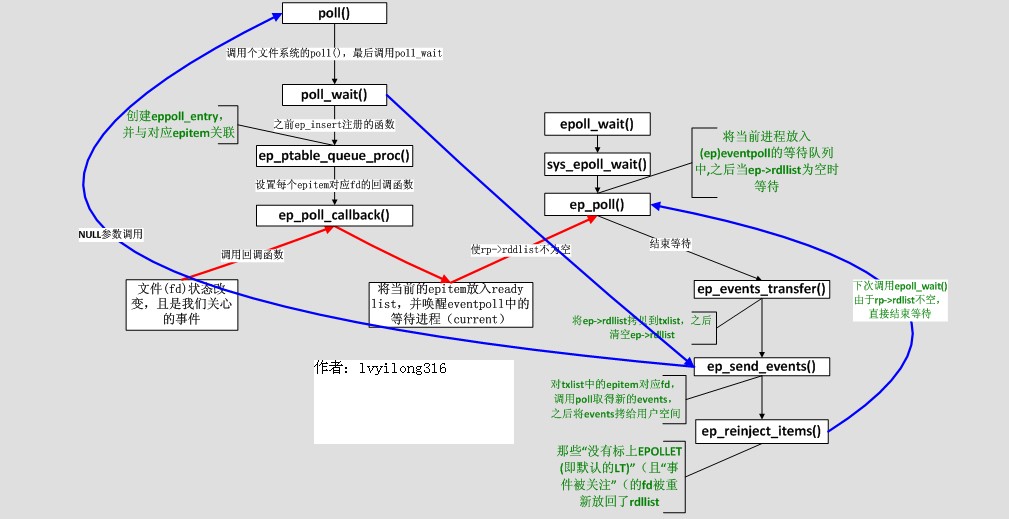
<https://blog.csdn.net/weiyuefei/article/details/52242778>

注：之前写过两篇关于epoll实现的文章，但是感觉懂得了实现原理并不一定会使用，所以又决定写这一系列文章，希望能够对epoll有比较清楚的认识。是请大家转载务必注明出处，算是对我劳动成果的一点点尊重吧。另外，文中如果有不全面或者不正确的地方还请大家指出。也可以私信或者发邮件：lvyilong316@163.com

**1. ET模式实现分析**

**1.1 ET和LT的实现区别**

    首先给出下面一张图，这张图是从我之前的一篇博文——epoll实现分析中摘取并细化的。这张图对理解ET模式已经epoll的工作过程只管重要，当然我自己总结出来后也感觉有的小成就，在这里与大家分享。



**注：上图的poll不要理解成和select相似那个poll，这是通过epoll\_ctl调用的。**

下面简要分析一下epoll的工作过程：

(1) epoll\_wait调用ep\_poll，当rdlist为空（无就绪fd）时挂起当前进程，知道rdlist不空时进程才被唤醒。

(2) 文件fd状态改变（buffer由不可读变为可读或由不可写变为可写），导致相应fd上的回调函数ep\_poll\_callback()被调用。

(3) ep\_poll\_callback将相应fd对应epitem加入rdlist，导致rdlist不空，进程被唤醒，epoll\_wait得以继续执行。

(4) ep\_events\_transfer函数将rdlist中的epitem拷贝到txlist中，并将rdlist清空。

(5) ep\_send\_events函数（很关键），它扫描**txlist中的每个epitem**，调用其关联fd对用的poll方法（图中蓝线）。此时对poll的调用仅仅是取得fd上较新的events（防止之前events被更新），之后将取得的events和相应的fd发送到用户空间（封装在struct epoll\_event，从epoll\_wait返回）。之后如果这个epitem对应的fd是LT模式监听且取得的events是用户所关心的，则将其重新加入回rdlist（图中蓝线），否则（ET模式）不在加入rdlist。

具体代码：

/\* 扫描整个txlist链表... \*/

for (eventcnt = 0, uevent = esed->events;

     !list\_empty(head) && eventcnt < esed->maxevents;) {

/\* 取出第一个成员 \*/

epi = list\_first\_entry(head, struct epitem, rdllink);

/\* 然后从链表里面移除 \*/

list\_del\_init(&epi->rdllink);

/\* 读取events,

 \* 注意events我们ep\_poll\_callback()里面已经取过一次了, 为啥还要再取?

 \* 1. 我们当然希望能拿到此刻的最新数据, events是会变的~

 \* 2. 不是所有的poll实现, 都通过等待队列传递了events, 有可能某些驱动压根没传

 \* 必须主动去读取. \*/

revents = epi->ffd.file->f\_op->poll(epi->ffd.file, NULL) &

epi->event.events;

if (revents) {

/\* 将当前的事件和用户传入的数据都copy给用户空间,

 \* 就是epoll\_wait()后应用程序能读到的那一堆数据. \*/

if (\_\_put\_user(revents, &uevent->events) ||

    \_\_put\_user(epi->event.data, &uevent->data)) {

/\* 如果copy过程中发生错误, 会中断链表的扫描,

 \* 并把当前发生错误的epitem重新插入到ready list.

 \* 剩下的没处理的epitem也不会丢弃, 在ep\_scan\_ready\_list()

 \* 中它们也会被重新插入到ready list \*/

list\_add(&epi->rdllink, head);

return eventcnt ? eventcnt : -EFAULT;

}

eventcnt++;

uevent++;

if (epi->event.events & EPOLLONESHOT)

epi->event.events &= EP\_PRIVATE\_BITS;

else if (!(epi->event.events & EPOLLET)) {

/\*

 \* If this file has been added with Level

 \* Trigger mode, we need to insert back inside

 \* the ready list, so that the next call to

 \* epoll\_wait() will check again the events

 \* availability. At this point, noone can insert

 \* into ep->rdllist besides us. The epoll\_ctl()

 \* callers are locked out by

 \* ep\_scan\_ready\_list() holding "mtx" and the

 \* poll callback will queue them in ep->ovflist.

 \*/

/\* 嘿嘿, EPOLLET和非ET的区别就在这一步之差呀~

 \* 如果是ET, epitem是不会再进入到readly list,

 \* 除非fd再次发生了状态改变, ep\_poll\_callback被调用.

 \* 如果是非ET, 不管你还有没有有效的事件或者数据,

 \* 都会被重新插入到ready list, 再下一次epoll\_wait

 \* 时, 会立即返回, 并通知给用户空间. 当然如果这个

 \* 被监听的fds确实没事件也没数据了, epoll\_wait会返回一个0,

 \* 空转一次.

 \*/

list\_add\_tail(&epi->rdllink, &ep->rdllist);

}

}

}

**说明：**

epoll\_wait返回的条件是rdlist不空，而使rdlist不空的途径有两个，分别对应图中的红线和蓝线。

ET和LT模式下的epitem都可以通过红线方式加入rdlist从而唤醒epoll\_wait，但LT模式下的epitem还可以通过蓝线方式重新加入rdlist唤醒epoll\_wait。所以ET模式下，fd就绪（通过红线加入rdlist）只会被通知一次，而LT模式下只要满足相应读写条件就返回就绪（通过蓝线加入rdlist）。

**ET事件发生仅通知一次的原因是只被添加到rdlist中一次，而LT可以有多次添加的机会。**

**1.2 两种加入rdlist途径的不同**

下面我们来分析一下图中两种将epitem加入rdlist方式（也就是红线和蓝线）的区别。

红线：**fd状态改变是才会触发。**那么什么情况会导致fd状态的改变呢？

对于读取操作：

(1) 当buffer由不可读状态变为可读的时候，即由空变为不空的时候。

(2) 当有新数据到达时，即buffer中的待读内容变多的时候。

对于写操作：

(1) 当buffer由不可写变为可写的时候，即由满状态变为不满状态的时候。

(2) 当有旧数据被发送走时，即buffer中待写的内容变少得时候。

蓝线：**fd的events中有相应的时间（位置1）即会触发。**那么什么情况下会改变events的相应位呢？

对于读操作：

(1) buffer中有数据可读的时候，即buffer不空的时候fd的events的可读为就置1。

对于写操作：

(1) buffer中有空间可写的时候，即buffer不满的时候fd的events的可写位就置1。

**说明：红线是时间驱动被动触发，蓝线是函数查询主动触发。**