<http://www.cnblogs.com/sduzh/p/6810469.html>

考虑如下情况（实际一般不会做，这里只是举个例子）：

1. 在主线程中创建一个socket、绑定到本地端口并监听
2. 在主线程中创建一个epoll实例(epoll\_create(2))
3. 将监听socket添加到epoll中(epoll\_ctl(2))
4. 创建多个子线程，每个子线程都共享步骤2里创建的同一个epoll文件描述符，然后调用epoll\_wait(2)等待事件到来accept(2)
5. 请求到来，新连接建立

这里的问题就是，在第5步的时候，会有多少个线程被唤醒而从epoll\_wait()调用返回？答案是不一定，可能只有一个，也可能有部分，也可能是全部。当然在多个线程都唤醒的情况下，只会有一个线程accept()调用会成功。

为何如此？从内核代码分析，原因如下：

在调用epoll\_wait(2)的时候，设置的epoll的等待队列回调函数是default\_wake\_function，添加队列的时候调用的是\_\_add\_wait\_queue\_exclusive()。

ep\_poll\_callback()中唤醒操作调用的是wake\_up\_locked(&ep->wq)，最终会调用\_\_wake\_up\_common，后者会判断exclusive标志：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | static void \_\_wake\_up\_common(wait\_queue\_head\_t \*q, unsigned int mode,              int nr\_exclusive, int wake\_flags, void \*key)  {      wait\_queue\_t \*curr, \*next;        list\_for\_each\_entry\_safe(curr, next, &q->task\_list, task\_list) {          unsigned flags = curr->flags;            if (curr->func(curr, mode, wake\_flags, key) &&                  (flags & WQ\_FLAG\_EXCLUSIVE) && !--nr\_exclusive)              break;      }  } |

因为\_\_wake\_up\_common()的调用是从wake\_up\_locked()开始的，\_\_wake\_up\_common的各个参数值为：

* q: struct eventpoll.wq
* mode： TASK\_NORMAL
* nr\_exclusive：1
* wake\_flags: 0
* key:NULL。

局部变量curr的值可以通过epoll\_wait()的源码得到，具体为：

* curr->flags: WQ\_FLAG\_EXCLUSIVE
* curr->func: default\_wake\_function

default\_wake\_function调用的是try\_to\_wake\_up。而try\_to\_wake\_up只有在要唤醒的进程状态不是TASK\_NORMAL时才会返回0，TASK\_NORMAL的定义是(TASK\_INTERRUPTIBLE | TASK\_UNINTERRUPTIBLE)。

因此\_\_wake\_up\_common里的if条件会在第一次判断的时候就满足，唤醒一个进程后便返回了，那为什么实际测试会发现有多个进程被唤醒呢？

原因就在于这个唯一被唤醒的进程。

当某个等待在epoll实例上的进程被唤醒后，最终会进入到[ep\_scan\_ready\_list](http://lxr.free-electrons.com/ident?i=ep_scan_ready_list)() 这个函数中，ep\_scan\_ready\_list()会以回调方式调用[ep\_send\_events\_proc](http://lxr.free-electrons.com/ident?i=ep_send_events_proc)()来将数据复制到用户空间。而ep\_scan\_ready\_list()函数在返回之前会再次判断epoll的就绪链表rdllist是否为空，如果不为空的话，就会再唤醒其他进程！下面就是ep\_scan\_ready\_list()返回之前的判断操作：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | if (!list\_empty(&ep->rdllist)) {      /\*       \* Wake up (if active) both the eventpoll wait list and       \* the ->poll() wait list (delayed after we release the lock).       \*/      if (waitqueue\_active(&ep->wq))          wake\_up\_locked(&ep->wq);      if (waitqueue\_active(&ep->poll\_wait))          pwake++;  } |

而在水平触发方式下，从就绪链表中移出来的文件描述符，如果当前仍有事件就绪（可读、可写等），会在复制到用户空间后被再次添加到就绪链表中：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | 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, no one 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.       \*/      list\_add\_tail(&epi->rdllink, &ep->rdllist);      ep\_pm\_stay\_awake(epi);  } |

因此在水平触发模式下，被唤醒的进程又会去唤醒其他进程，除非当前事件已经被处理完或者所有进程都已经被唤醒（被唤醒的进程会从epoll等待队列上移除）。