<https://blog.csdn.net/weiyuefei/article/details/52242872>

首先，看程序四的例子。

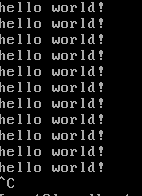
程序四

点击(此处)折叠或打开

1. #include <unistd.h>
2. #include <iostream>
3. #include <sys/epoll.h>
4. using namespace std;
5. int main(void)
6. {
7. int epfd,nfds;
8. struct epoll\_event ev,events[5];//ev用于注册事件，数组用于返回要处理的事件
9. epfd=epoll\_create(1);//只需要监听一个描述符——标准输出
10. ev.data.fd=STDOUT\_FILENO;
11. ev.events=EPOLLOUT|EPOLLET;//监听读状态同时设置ET模式
12. epoll\_ctl(epfd,EPOLL\_CTL\_ADD,STDOUT\_FILENO,&ev);//注册epoll事件
13. for(;;)
14. {
15. nfds=epoll\_wait(epfd,events,5,-1);
16. for(int i=0;i<nfds;i++)
17. {
18. if(events[i].data.fd==STDOUT\_FILENO)
19. cout<<"hello world!"<<endl;
20. }
21. }
22. };

这个程序的功能是只要标准输出写就绪，就输出“hello world！”。

运行结果：



我们发现这将是一个死循环。下面具体分析一下这个程序的执行过程：

(1) 首先初始buffer为空，buffer中有空间可写，这时无论是ET还是LT都会将对应的epitem加入rdlist(对应第一节图中的红线)，导致epoll\_wait就返回写就绪。

(2) 程序想标准输出输出”hello world！”和换行符，因为标准输出为控制台的时候缓冲是“行缓冲”,所以换行符导致buffer中的内容清空，这就对应第二节中ET模式下写就绪的第二种情况——**当有旧数据被发送走时，即buffer中待写的内容变少得时候会触发fd状态的改变。**所以下次epoll\_wait会返回写就绪。之后重复这个过程一直循环下去。

我们再看程序五。

程序五

相对程序四这里仅仅去掉了输出的换行操作。即：

**cout<<"hello world!";**

运行结果如下：

http://blog.chinaunix.net/attachment/201406/9/28541347_1402329574ZU6X.jpg

我们看到程序成挂起状态。因为第一次epoll\_wait返回写就绪后，程序向标准输出的buffer中写入“hello world!”，但是因为没有输出换行，所以buffer中的内容一直存在，下次epoll\_wait的时候，虽然有写空间但是ET模式下不再返回写就绪。回忆第一节关于ET的实现，这种情况原因就是第一次buffer为空，导致epitem加入rdlist，返回一次就绪后移除此epitem，之后虽然buffer仍然可写，但是由于对应epitem已经不再rdlist中，就不会对其就绪fd的events的在检测了。

程序六

点击(此处)折叠或打开

1. int main(void)
2. {
3. int epfd,nfds;
4. struct epoll\_event ev,events[5];//ev用于注册事件，数组用于返回要处理的事件
5. epfd=epoll\_create(1);//只需要监听一个描述符——标准输出
6. ev.data.fd=STDOUT\_FILENO;
7. ev.events=EPOLLOUT;//使用默认LT模式
8. epoll\_ctl(epfd,EPOLL\_CTL\_ADD,STDOUT\_FILENO,&ev);//注册epoll事件
9. for(;;)
10. {
11. nfds=epoll\_wait(epfd,events,5,-1);
12. for(int i=0;i<nfds;i++)
13. {
14. if(events[i].data.fd==STDOUT\_FILENO)
15. cout<<"hello world!";
16. }
17. }
18. };

http://blog.chinaunix.net/attachment/201406/10/28541347_14023297191ITy.png

程序六相对程序五仅仅是修改ET模式为默认的LT模式，我们发现程序再次死循环。这时候原因已经很清楚了，因为当向buffer写入”hello world!”后，虽然buffer没有输出清空，但是LT模式下只有buffer有写空间就返回写就绪，所以会一直输出”hello world!”,当buffer满的时候，buffer会自动刷清输出，同样会造成epoll\_wait返回写就绪。

程序七

点击(此处)折叠或打开

1. int main(void)
2. {
3. int epfd,nfds;
4. struct epoll\_event ev,events[5];//ev用于注册事件，数组用于返回要处理的事件
5. epfd=epoll\_create(1);//只需要监听一个描述符——标准输出
6. ev.data.fd=STDOUT\_FILENO;
7. ev.events=EPOLLOUT|EPOLLET;//监听读状态同时设置ET模式
8. epoll\_ctl(epfd,EPOLL\_CTL\_ADD,STDOUT\_FILENO,&ev);//注册epoll事件
9. for(;;)
10. {
11. nfds=epoll\_wait(epfd,events,5,-1);
12. for(int i=0;i<nfds;i++)
13. {
14. if(events[i].data.fd==STDOUT\_FILENO)
15. cout<<"hello world!";
16. ev.data.fd=STDOUT\_FILENO;
17. ev.events=EPOLLOUT|EPOLLET;
18. epoll\_ctl(epfd,EPOLL\_CTL\_MOD,STDOUT\_FILENO,&ev); //重新MOD事件（ADD无效）
19. }
20. }
21. };

http://blog.chinaunix.net/attachment/201406/10/28541347_1402329859x5vl.png

程序七相对于程序五在每次向标准输出的buffer输出”hello world!”后，重新MOD OUT事件。所以相当于每次重新进行第一节中红线描述的途径返回就绪，导致程序循环输出。