<https://blog.csdn.net/weiyuefei/article/details/52242867>

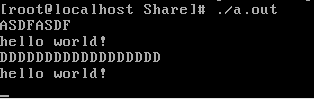
首先看程序一，这个程序想要实现的功能是当用户从控制台有任何输入操作时，输出”hello world！”。

程序一

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——标准输入
10. ev.data.fd=STDIN\_FILENO;
11. ev.events=EPOLLIN|EPOLLET;//监听读状态同时设置ET模式
12. epoll\_ctl(epfd,EPOLL\_CTL\_ADD,STDIN\_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==STDIN\_FILENO)
19. cout<<"hello world!"<<endl;
20. }
21. }
22. }

运行结果：



程序一中对标准输入的监听使用ET模式，结果实现了我们想要的功能。那么实际原理是如何呢，我们将过程分析一下：

(1) 当用户输入一组字符，这组字符被送入buffer，字符停留在buffer中，又因为buffer由空变为不空，所以ET返回读就绪，输出”hello world！”。

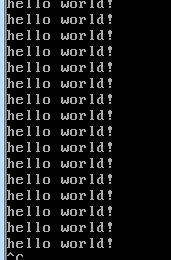
(2) 之后程序再次执行epoll\_wait，此时虽然buffer中有内容可读，但是根据我们上节的分析，ET并不返回就绪，导致epoll\_wait阻塞。（底层原因是ET下就绪fd的epitem只被放入rdlist一次）。

(3) 用户再次输入一组字符，导致buffer中的内容增多，根据我们上节的分析这将导致fd状态的改变，是对应的epitem再次加入rdlist，从而使epoll\_wait返回读就绪，再次输出“hello world！”。

 我们在看看LT的情况如何，将程序一以下修改：

**ev.events=EPOLLIN;**//默认使用LT模式

运行结果：



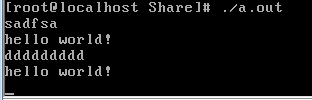
结果正如我们所料，程序出现死循环，因为用户输入任意数据后，数据被送入buffer且没有被读出，所以LT模式下每次epoll\_wait都认为buffer可读返回读就绪。导致每次都会输出”hello world！”。下面在看程序二。

程序二

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. char buf[256];
9. 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
10. epfd=epoll\_create(1);//只需要监听一个描述符——标准输入
11. ev.data.fd=STDIN\_FILENO;
12. ev.events=EPOLLIN;//使用默认LT模式
13. epoll\_ctl(epfd,EPOLL\_CTL\_ADD,STDIN\_FILENO,&ev);//注册epoll事件
14. for(;;)
15. {
16. nfds=epoll\_wait(epfd,events,5,-1);
17. for(int i=0;i<nfds;i++)
18. {
19. if(events[i].data.fd==STDIN\_FILENO)
20. {
21. read(STDIN\_FILENO,buf,sizeof(buf));//将缓冲中的内容读出
22. cout<<"hello world!"<<endl;
23. }
24. }
25. }
26. }

运行结果：



程序二依然使用LT模式，但是每次epoll\_wait返回读就绪的时候我们都将buffer（缓冲）中的内容read出来，所以导致buffer再次清空，下次调用epoll\_wait就会阻塞。所以能够实现我们所想要的功能——当用户从控制台有任何输入操作时，输出”hello world！”。我们再来看看程序三。

程序三

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——标准输入
6. ev.data.fd=STDIN\_FILENO;
7. ev.events=EPOLLIN|EPOLLET;//使用默认LT模式
8. epoll\_ctl(epfd,EPOLL\_CTL\_ADD,STDIN\_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==STDIN\_FILENO)
15. {
16. cout<<"hello world!"<<endl;
17. ev.data.fd=STDIN\_FILENO;
18. ev.events=EPOLLIN|EPOLLET;//使用默认LT模式
19. epoll\_ctl(epfd,EPOLL\_CTL\_MOD,STDIN\_FILENO,&ev);//重新MOD事件（ADD无效）
20. }
21. }
22. }
23. }



程序三依然使用ET，但是每次读就绪后都主动的再次MOD IN事件，我们发现程序再次出现死循环，也就是每次返回读就绪。这就验证了上一节讨论ET读就绪的第三种情况。但是注意，如果我们将MOD改为ADD，将不会产生任何影响。别忘了每次ADD一个描述符都会在epitem组成的红黑树中添加一个项，我们之前已经ADD过一次，再次ADD将阻止添加，所以在次调用ADD IN事件不会有任何影响。