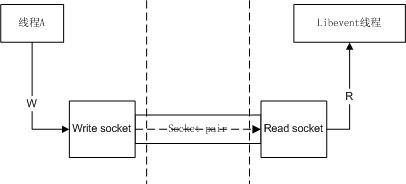
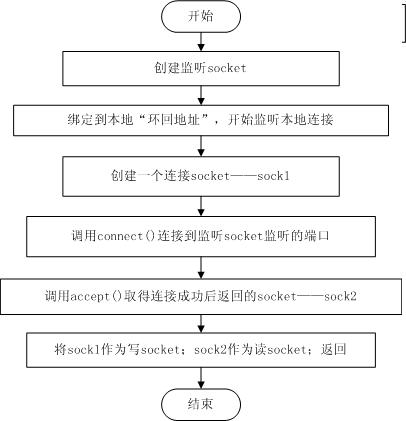
**libevent源码深度剖析八**

——集成信号处理  
张亮

      现在我们已经了解了libevent的基本框架：事件管理框架和事件主循环。上节提到了libevent中I/O事件和Signal以及Timer事件的集成，这一节将分析如何将Signal集成到事件主循环的框架中。

**1 集成策略——使用socket pair**

      前一节已经做了足够多的介绍了，基本方法就是采用“消息机制”。在libevent中这是通过socket pair完成的，下面就来详细分析一下。  
      Socket pair就是一个socket对，包含两个socket，一个读socket，一个写socket。工作方式如下图所示：  
                                   
  
    创建一个socket pair并不是复杂的操作，可以参见下面的流程图，清晰起见，其中忽略了一些错误处理和检查。

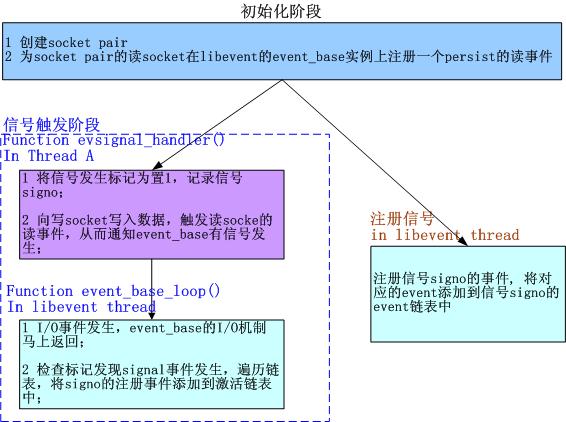
   
                 
Libevent提供了辅助函数evutil\_socketpair()来创建一个socket pair，可以结合上面的创建流程来分析该函数。

**2 集成到事件主循环——通知event\_base**

      Socket pair创建好了，可是libevent的事件主循环还是不知道Signal是否发生了啊，看来我们还差了最后一步，那就是：为socket pair的读socket在libevent的event\_base实例上注册一个persist的读事件。  
      这样当向写socket写入数据时，读socket就会得到通知，触发读事件，从而event\_base就能相应的得到通知了。  
前面提到过，Libevent会在事件主循环中检查标记，来确定是否有触发的signal，如果标记被设置就处理这些signal，这段代码在各个具体的I/O机制中，以Epoll为例，在epoll\_dispatch()函数中，代码片段如下：

**[cpp]** [view plain](http://blog.csdn.net/sparkliang/article/details/5011400) [copy](http://blog.csdn.net/sparkliang/article/details/5011400)

1. res = epoll\_wait(epollop->epfd, events, epollop->nevents, timeout);
2. **if** (res == -1) {
3. **if** (errno != EINTR) {
4. event\_warn("epoll\_wait");
5. **return** (-1);
6. }
7. evsignal\_process(base);// 处理signal事件
8. **return** (0);
9. } **else** **if** (base->sig.evsignal\_caught) {
10. evsignal\_process(base);// 处理signal事件
11. }

完整的处理框架如下所示：  
             
注1：libevent中，初始化阶段并不注册读socket的读事件，而是在注册信号阶段才会**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "_blank)**并注册；  
注2：libevent中，检查I/O事件是在各系统I/O机制的dispatch()函数中完成的，该dispatch()函数在event\_base\_loop()函数中被调用；

**3 evsignal\_info结构体**

      Libevent中Signal事件的管理是通过结构体evsignal\_info完成的，结构体位于evsignal.h文件中，定义如下：

**[cpp]** [view plain](http://blog.csdn.net/sparkliang/article/details/5011400) [copy](http://blog.csdn.net/sparkliang/article/details/5011400)

1. **struct** evsignal\_info {
2. **struct** event ev\_signal;
3. **int** ev\_signal\_pair[2];
4. **int** ev\_signal\_added;
5. **volatile** **sig\_atomic\_t** evsignal\_caught;
6. **struct** event\_list evsigevents[NSIG];
7. **sig\_atomic\_t** evsigcaught[NSIG];
8. #ifdef HAVE\_SIGACTION
9. **struct** sigaction \*\*sh\_old;
10. #else
11. ev\_sighandler\_t \*\*sh\_old;
12. #endif
13. **int** sh\_old\_max;
14. };

下面详细介绍一下个字段的含义和作用：  
1）ev\_signal， 为socket pair的读socket向event\_base注册读事件时使用的event结构体；  
2）ev\_signal\_pair，socket pair对，作用见第一节的介绍；  
3）ev\_signal\_added，记录ev\_signal事件是否已经注册了；  
4）evsignal\_caught，是否有信号发生的标记；是volatile类型，因为它会在另外的线程中被修改；  
5）evsigvents[NSIG]，数组，evsigevents[signo]表示注册到信号signo的事件链表；  
6）evsigcaught[NSIG]，具体记录每个信号触发的次数，evsigcaught[signo]是记录信号signo被触发的次数；  
7）sh\_old记录了原来的signal处理函数指针，当信号signo注册的event被清空时，需要重新设置其处理函数；  
    evsignal\_info的初始化包括，创建socket pair，设置ev\_signal事件（但并没有注册，而是等到有信号注册时才检查并注册），并将所有标记置零，初始化信号的注册事件链表指针等。

**4 注册、注销signal事件**

      注册signal事件是通过evsignal\_add(struct event \*ev)函数完成的，libevent对所有的信号注册同一个处理函数evsignal\_handler()，该函数将在下一段介绍，注册过程如下：  
1 取得ev要注册到的信号signo；  
2 如果信号signo未被注册，那么就为signo注册信号处理函数evsignal\_handler()；  
3 如果事件ev\_signal还没哟注册，就注册ev\_signal事件；  
4 将事件ev添加到signo的event链表中；  
从signo上注销一个已注册的signal事件就更简单了，直接从其已注册事件的链表中移除即可。如果事件链表已空，那么就恢复旧的处理函数；  
下面的讲解都以signal()函数为例，sigaction()函数的处理和signal()相似。  
处理函数evsignal\_handler()函数做的事情很简单，就是记录信号的发生次数，并通知event\_base有信号触发，需要处理：

**[cpp]** [view plain](http://blog.csdn.net/sparkliang/article/details/5011400) [copy](http://blog.csdn.net/sparkliang/article/details/5011400)

1. **static** **void** evsignal\_handler(**int** sig)
2. {
3. **int** save\_errno = errno; // 不覆盖原来的错误代码
4. **if** (evsignal\_base == NULL) {
5. event\_warn("%s: received signal %d, but have no base configured", \_\_func\_\_, sig);
6. **return**;
7. }
8. // 记录信号sig的触发次数，并设置event触发标记
9. evsignal\_base->sig.evsigcaught[sig]++;
10. evsignal\_base->sig.evsignal\_caught = 1;
11. #ifndef HAVE\_SIGACTION
12. signal(sig, evsignal\_handler); // 重新注册信号
13. #endif
14. // 向写socket写一个字节数据，触发event\_base的I/O事件，从而通知其有信号触发，需要处理
15. send(evsignal\_base->sig.ev\_signal\_pair[0], "a", 1, 0);
16. errno = save\_errno; // 错误代码
17. }

5 小节  
本节介绍了libevent对signal事件的具体处理框架，包括事件注册、删除和socket pair通知机制，以及是如何将Signal事件集成到事件主循环之中的。