<https://blog.csdn.net/luotuo44/article/details/38424173>

**转载请注明出处:**[**http://blog.csdn.net/luotuo44/article/details/38424173**](http://blog.csdn.net/luotuo44/article/details/38424173)

# 相关结构体：

        因为event\_signal\_map结构体实在太简单了，所以不像event\_io\_map那样，有一个专门的文件。由于没有专门的文件，那么只能从蛛丝马迹上探索这个event\_signal\_map结构了。

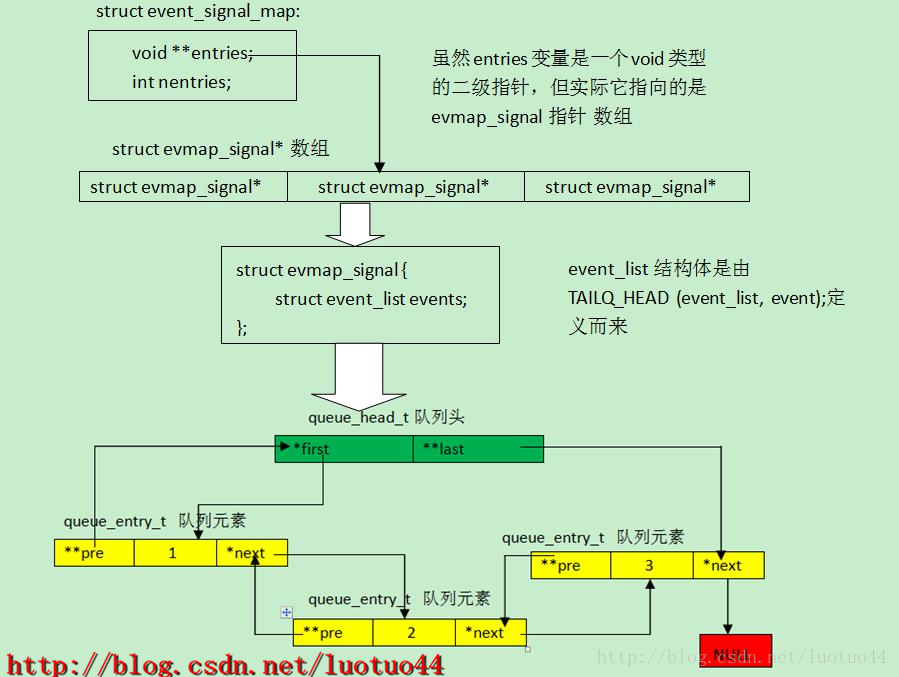
        通过一些搜索，可以得到与event\_signal\_map相关联的一些结构体有下面这些：

**[cpp]** [view plain](https://blog.csdn.net/luotuo44/article/details/38424173) [copy](https://blog.csdn.net/luotuo44/article/details/38424173)

1. //TAILQ\_HEAD (event\_list, event); event\_struct.h
2. **struct** event\_list
3. {
4. **struct** event \*tqh\_first;
5. **struct** event \*\*tqh\_last;
6. };
8. **struct** evmap\_signal {
9. **struct** event\_list events;
10. };
12. **struct** event\_signal\_map {
13. /\* An array of evmap\_io \* or of evmap\_signal \*; empty entries are
14. \* set to NULL. \*/
15. **void** \*\*entries; //二级指针，evmap\_signal\*数组
16. **int** nentries; //元素个数
17. };

        相对于event\_io\_map是一个哈希表，event\_signal\_map是一个数组。并且entries虽然是void\*\*，但实际它是一个structevmap\_signal指针数组。而evmap\_signal成员只有一个TAILQ\_HEAD (event\_list, event);

        由上面这些结构配合得到的结构如下所示：



        从上图可以看到，event\_signal\_map结构体指向了一个evmap\_signal指针数组，而evmap\_signal结构体可以说就是一个TAILQ\_QUEUE队列的队列头。通过TAILQ\_QUEUE队列和图中的结构，event\_signal\_map就可以把event结构体都关联并管理起来。

        上图中会有一个event结构体队列，原因和[前一篇博文](http://blog.csdn.net/luotuo44/article/details/38403241)是一样的。因为同一个文件描述符fd或者信号值sig是可以多次调用event\_new、event\_add函数的。即存在一对多。这个队列就是把同一个fd或者sig的event连起来，方便管理。

# 操作函数：

        虽然从上图看到event\_signal\_map结构是比较简单，现在还是要看一下与event\_signal\_map相关联的函数。先看一个内存分配函数。

**[cpp]** [view plain](https://blog.csdn.net/luotuo44/article/details/38424173) [copy](https://blog.csdn.net/luotuo44/article/details/38424173)

1. //evmap.c文件
2. //slot是信号值sig，或者文件描述符fd.
3. //当sig或者fd >= map的nentries变量时就会调用此函数
4. **static** **int**   //msize 等于 sizeof(struct evmap\_signal \*)
5. evmap\_make\_space(**struct** event\_signal\_map \*map, **int** slot, **int** msize)
6. {
7. **if** (map->nentries <= slot) {
8. //posix标准中，信号的种类就只有32种。
9. //http://blog.csdn.net/luotuo44/article/details/16799607
10. //在Windows中，信号的种类就更少了，只有6种。
11. //http://msdn.microsoft.com/zh-cn/library/xdkz3x12.aspx
12. //所以一开始取32还是比较合理的
13. **int** nentries = map->nentries ? map->nentries : 32;
14. **void** \*\*tmp;
16. //当slot是一个文件描述符时，就会大于32
17. **while** (nentries <= slot)
18. nentries <<= 1;
20. tmp = (**void** \*\*)mm\_realloc(map->entries, nentries \* msize);
21. **if** (tmp == NULL)
22. **return** (-1);
24. //清零是很有必要的。因为tmp是二级指针，数组里面的元素是一个指针
25. memset(&tmp[map->nentries], 0,
26. (nentries - map->nentries) \* msize);
28. map->nentries = nentries;
29. map->entries = tmp;
30. }
32. **return** (0);
33. }

        在非Windows系统中，event\_io\_map被定义成event\_signal\_map,此时slot将是文件描述符fd。但对于这些遵循POSIX标准的OS来说，fd都是从0递增的比较小的值。所以代码中的while( nentrie <= slot ) nentries <<= 1; 并不会很大。对于slot为一个信号值，最大也就只有32。所以总得来说，nentries的值并不会太大。

        从上面的代码也可以想到，对于参数slot，它要么是信号值sig，要么是文件描述符fd。而event\_signal\_map要求的数组长度一定要大于slot。那么之后给定一个sig或者fd，就可以直接通过下标操作快速定位了。这是因为一个sig或者fd就对应在数组中占有一个位置，并且sig或者fd的值等于其在数组位置的下标值。这种方式无论是管理还是代码复杂度都要比[前一篇博文](http://blog.csdn.net/luotuo44/article/details/38403241)说到的哈希表要简单。

# 在Libevent中的应用：

        同event\_io\_map一样，event\_signal\_map同样也是有event\_signal\_add函数的。不过后者比前者简单了很多。

**[cpp]** [view plain](https://blog.csdn.net/luotuo44/article/details/38424173) [copy](https://blog.csdn.net/luotuo44/article/details/38424173)

1. //event.c文件
2. **int**
3. evmap\_signal\_add(**struct** event\_base \*base, **int** sig, **struct** event \*ev)
4. {
5. **const** **struct** eventop \*evsel = base->evsigsel;
6. **struct** event\_signal\_map \*map = &base->sigmap;
7. **struct** evmap\_signal \*ctx = NULL;
9. **if** (sig >= map->nentries) {
11. **if** (evmap\_make\_space(map, sig, **sizeof**(**struct** evmap\_signal \*)) == -1)
12. **return** (-1);
13. }
15. //无论是GET\_SIGNAL\_SLOT\_AND\_CTOR还是GET\_IO\_SLOT\_AND\_CTOR，其作用
16. //都是在数组(哈希表也是一个数组)中找到fd中的一个结构。
17. //GET\_SIGNAL\_SLOT\_AND\_CTOR(ctx, map, sig, evmap\_signal, evmap\_signal\_init,
18. //base->evsigsel->fdinfo\_len);
19. **do**
20. {
21. //同event\_io\_map一样，同一个信号或者fd可以被多次event\_new、event\_add
22. //所以，当同一个信号或者fd被多次event\_add后，entries[sig]就不会为NULL
23. **if** ((map)->entries[sig] == NULL)//第一次
24. {
25. //evmap\_signal成员只有一个TAILQ\_HEAD (event\_list, event);
26. //可以说evmap\_signal本身就是一个TAILQ\_HEAD
27. //这个赋值操作很重要。
28. (map)->entries[sig] = mm\_calloc(1, **sizeof**(**struct** evmap\_signal)
29. + base->evsigsel->fdinfo\_len
30. );
32. **if** (EVUTIL\_UNLIKELY((map)->entries[sig] == NULL))
33. **return** (-1);

36. //内部调用TAILQ\_INIT(&entry->events);
37. (evmap\_signal\_init)((**struct** evmap\_signal \*)(map)->entries[sig]);
38. }
40. (ctx) = (**struct** evmap\_signal \*)((map)->entries[sig]);
41. } **while** (0);
43. ...
45. //将所有有相同信号值的event连起来
46. TAILQ\_INSERT\_TAIL(&ctx->events, ev, ev\_signal\_next);
48. **return** (1);
49. }

        同样，GET\_SIGNAL\_SLOT\_AND\_CTOR宏的作用就是让ctx指向structevmap\_signal结构体中的TAILQ\_HEAD。这样就可以使用TAILQ\_INSERT\_TAIL宏，把ev变量插入到队列中。如果有现成的struct evmap\_signal就直接使用，没有的话就新建一个。

        有一点要提出，虽然[前一篇博文](http://blog.csdn.net/luotuo44/article/details/38403241)提到在非Windows系统中，event\_io\_map也被定义成了event\_signal\_map，但实际上他们并不会由链表连在一起。因为在event\_base结构体中，分别有struct event\_io\_map变量io和event\_signal\_map变量sigmap。所有的io类型的event结构体会被放到io中，而信号类型的event则会被放到sigmap变量中。

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/luotuo44/article/details/38424173