event\_base\_new()函数

每次用libevent时，要做的第一件事就是initialize。如果用event\_init()函数，它会在内部调用event\_base\_new()函数，所以说libevent初始化的核心就是event\_base\_new()函数，那么这个函数内部都需要做什么工作呢?

首先来看一下这个函数的实现:

struct event\_base \*

event\_base\_new(void)

{

int i;

struct event\_base \*base;

if ((base = calloc(1, sizeof(struct event\_base))) == NULL)

event\_err(1, "%s: calloc", \_\_func\_\_);

detect\_monotonic();

gettime(base, &base->event\_tv);

min\_heap\_ctor(&base->timeheap);

TAILQ\_INIT(&base->eventqueue);

base->sig.ev\_signal\_pair[0] = -1;

base->sig.ev\_signal\_pair[1] = -1;

//根据系统配置和编译选项决定使用哪一种I/O demultiplex机制

//可以看出,libevent在编译阶段选择系统的I/O demultiplex机制，而不支持在运行阶段根据配置再次选择

base->evbase = NULL; //赋为NULL，相当于找到之后break

for (i = 0; eventops[i] && !base->evbase; i++) { //按顺序遍历，因为这些机制是按性能由大到小排的

base->evsel = eventops[i];

base->evbase = base->evsel->init(base);

}

if (base->evbase == NULL)

event\_errx(1, "%s: no event mechanism available", \_\_func\_\_);

if (evutil\_getenv("EVENT\_SHOW\_METHOD"))

event\_msgx("libevent using: %s\n",

base->evsel->name);

/\* allocate a single active event queue \*/

event\_base\_priority\_init(base, 1);

return (base);

}

我们来详细说明一下该函数都做了哪些初始化的工作。

1. 分配一个event\_base结构体。我们可以称它为libevent的反应堆，函数返回的时候将返回该反应堆的指针，我们用它来管理各种事件。本函数中会对该反应堆成员进行初始化。
2. 检测是否支持monotonic时间。monotonic时间是这样子的：

**CLOCK\_MONOTONIC**

代表从过去某个固定的时间点开始的绝对的逝去时间，它不受任何系统time-of-day时钟修改的影响，如果你想计算出在一台计算机上不受重启的影响，两个事件发生的间隔时间的话，那么它将是最好的选择。（与之相对的CLOCK\_REALTIME时间，即当前的我们所常看的时间)

1. 初始化base->event\_tv。gettime()中会检测是否有时间tv\_cache缓存，如果有就直接用，没有，就用clock\_gettime()函数获取系统monotonic时间。
2. 最小堆构造。min\_heap\_ctor()是最小堆的构造函数，该构造函数会将最小堆成员进行零初始化。
3. 已注册事件队列初始化。TAILQ\_INIT是/sys/queue.h中的针对tail queue的初始化函数，调用它之后会初始化tail queue头结点。
4. socket pair初始化。全部初始化为-1。socket pair用来实现signal事件和I/O事件的统一。
5. 选择并初始化I/O复用机制。这个比较重要，不用多说。
6. 初始化活跃事件队列。event\_base\_priority\_init()函数负责实现该功能。

(author:WY.Huang 15702937529@163.com)