**event\_new创建事件**

使用event\_new() 创建事件

使用event\_free() 释放事件

#define EV\_TIMEOUT 0x01

#define EV\_READ 0x02

#define EV\_WRITE 0x04

#define EV\_SIGNAL 0x08

#define EV\_PERSIST 0x10

#define EV\_ET 0x20

typedef void (\*event\_callback\_fn)(evutil\_socket\_t fd, short what, void \*);

struct event \*event\_new(struct event\_base \*base, evutil\_socket\_t fd,

short what, event\_callback\_fn cb,

void \*arg);

void event\_free(struct event \*event);

fd：要监听的文件描述符

what：上述标志的集合，表示要监听的事件（如可读可写）

EV\_TIMEOUT：这个标志表示某超时时间流逝后事件成为激活的

EV\_READ：可读

EV\_WRITE：可写

EV\_SIGNAL：基于信号的事件处理，指定fd是一个信号，而不是文件描述符

EV\_PERSIST：表示事件是“持久的”，如果不添加此标志，事件被触发后（如可读或可写），event将从event\_base中移除

EV\_ET：如果底层的 event\_base 后端支持边沿触发事件,则事件应该是边沿触发的

cb：事件发生时调用的回调函数

arg：回调函数被调用是，作为回调函数的第三个参数

**evsignal\_new创建信号事件**

Libevent提供了信号事件的生成

#define evsignal\_new(base, signum, cb, arg) \

event\_new(base, signum, EV\_SIGNAL|EV\_PERSIST, cb, arg)

**设置事件优先级**

int event\_priority\_set(struct event \*event, int priority);

priority：

取值在 0和 event\_base 的优先级数目减去1之间的数值，多个优先级的事件同时被激活时，高优先级的会先运行，默认的优先级是 event\_base 的优先级数目除以2

返回值：成功返回0，失败返回1

**event\_add：向event\_base添加event**

event\_add：向event\_base添加event

event\_del：向event\_base移除event

int event\_add(struct event \*ev, const struct timeval \*tv);

int event\_del(struct event \*ev);

ev：事件处理器

tv：超时值，NULL为添加的事件不会超时

**示例**

#include <event2/event.h>

void cb\_func(evutil\_socket\_t fd, short what, void \*arg){

const char \*data = arg;

printf("Got an event on socket %d:%s%s%s%s [%s]",

(int) fd,

(what&EV\_TIMEOUT) ? " timeout" : "",

(what&EV\_READ) ? " read" : "",

(what&EV\_WRITE) ? " write" : "",

(what&EV\_SIGNAL) ? " signal" : "",

data);

}

void main\_loop(evutil\_socket\_t fd1, evutil\_socket\_t fd2){

struct event \*ev1, \*ev2;

struct timeval five\_seconds = {5,0};

struct event\_base \*base = event\_base\_new();

/\* The caller has already set up fd1, fd2 somehow, and make them

nonblocking. \*/

ev1 = event\_new(base, fd1, EV\_TIMEOUT|EV\_READ|EV\_PERSIST, cb\_func,

(char\*)"Reading event");

ev2 = event\_new(base, fd2, EV\_WRITE|EV\_PERSIST, cb\_func,

(char\*)"Writing event");

event\_add(ev1, &five\_seconds);

event\_add(ev2, NULL);

event\_base\_dispatch(base);

}