<https://www.cnblogs.com/wainiwann/p/7096245.html>

**转载请注明出处：**[**http://blog.csdn.net/luotuo44/article/details/39670221**](http://blog.csdn.net/luotuo44/article/details/39670221)

        本文从简单到复杂，展示如何使用libevent。网上的许多例子都是只有服务器端的，本文里面客户端和服务器端都有，以飨读者。

        关于libevent编程时的一些疑问可以阅读《[libevent编程疑难解答](http://blog.csdn.net/luotuo44/article/details/39547391)》。假如读者还想了解libevent的具体实现，可以阅读《[libevent源码分析](http://blog.csdn.net/luotuo44/article/category/2435521)》系统文章。

        不说这么多了，直接上代码。

# 初等：

## 客户端代码：

[复制代码](javascript:void(0);)

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<errno.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<event.h>

#include<event2/util.h>

int tcp\_connect\_server(const char\* server\_ip, int port);

void cmd\_msg\_cb(int fd, short events, void\* arg);

void socket\_read\_cb(int fd, short events, void \*arg);

int main(int argc, char\*\* argv)

{

if( argc < 3 )

{

printf("please input 2 parameter\n");

return -1;

}

//两个参数依次是服务器端的IP地址、端口号

int sockfd = tcp\_connect\_server(argv[1], atoi(argv[2]));

if( sockfd == -1)

{

perror("tcp\_connect error ");

return -1;

}

printf("connect to server successful\n");

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

struct event \*ev\_sockfd = event\_new(base, sockfd,

EV\_READ | EV\_PERSIST,

socket\_read\_cb, NULL);

event\_add(ev\_sockfd, NULL);

//监听终端输入事件

struct event\* ev\_cmd = event\_new(base, STDIN\_FILENO,

EV\_READ | EV\_PERSIST, cmd\_msg\_cb,

(void\*)&sockfd);

event\_add(ev\_cmd, NULL);

event\_base\_dispatch(base);

printf("finished \n");

return 0;

}

void cmd\_msg\_cb(int fd, short events, void\* arg)

{

char msg[1024];

int ret = read(fd, msg, sizeof(msg));

if( ret <= 0 )

{

perror("read fail ");

exit(1);

}

int sockfd = \*((int\*)arg);

//把终端的消息发送给服务器端

//为了简单起见，不考虑写一半数据的情况

write(sockfd, msg, ret);

}

void socket\_read\_cb(int fd, short events, void \*arg)

{

char msg[1024];

//为了简单起见，不考虑读一半数据的情况

int len = read(fd, msg, sizeof(msg)-1);

if( len <= 0 )

{

perror("read fail ");

exit(1);

}

msg[len] = '\0';

printf("recv %s from server\n", msg);

}

typedef struct sockaddr SA;

int tcp\_connect\_server(const char\* server\_ip, int port)

{

int sockfd, status, save\_errno;

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr) );

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(port);

status = inet\_aton(server\_ip, &server\_addr.sin\_addr);

if( status == 0 ) //the server\_ip is not valid value

{

errno = EINVAL;

return -1;

}

sockfd = ::socket(PF\_INET, SOCK\_STREAM, 0);

if( sockfd == -1 )

return sockfd;

status = ::connect(sockfd, (SA\*)&server\_addr, sizeof(server\_addr) );

if( status == -1 )

{

save\_errno = errno;

::close(sockfd);

errno = save\_errno; //the close may be error

return -1;

}

evutil\_make\_socket\_nonblocking(sockfd);

return sockfd;

}

[复制代码](javascript:void(0);)

## 服务器端代码：

[复制代码](javascript:void(0);)

#include<stdio.h>

#include<string.h>

#include<errno.h>

#include<unistd.h>

#include<event.h>

void accept\_cb(int fd, short events, void\* arg);

void socket\_read\_cb(int fd, short events, void \*arg);

int tcp\_server\_init(int port, int listen\_num);

int main(int argc, char\*\* argv)

{

int listener = tcp\_server\_init(9999, 10);

if( listener == -1 )

{

perror(" tcp\_server\_init error ");

return -1;

}

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

//添加监听客户端请求连接事件

struct event\* ev\_listen = event\_new(base, listener, EV\_READ | EV\_PERSIST,

accept\_cb, base);

event\_add(ev\_listen, NULL);

event\_base\_dispatch(base);

return 0;

}

void accept\_cb(int fd, short events, void\* arg)

{

evutil\_socket\_t sockfd;

struct sockaddr\_in client;

socklen\_t len = sizeof(client);

sockfd = ::accept(fd, (struct sockaddr\*)&client, &len );

evutil\_make\_socket\_nonblocking(sockfd);

printf("accept a client %d\n", sockfd);

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

//仅仅是为了动态创建一个event结构体

struct event \*ev = event\_new(NULL, -1, 0, NULL, NULL);

//将动态创建的结构体作为event的回调参数

event\_assign(ev, base, sockfd, EV\_READ | EV\_PERSIST,

socket\_read\_cb, (void\*)ev);

event\_add(ev, NULL);

}

void socket\_read\_cb(int fd, short events, void \*arg)

{

char msg[4096];

struct event \*ev = (struct event\*)arg;

int len = read(fd, msg, sizeof(msg) - 1);

if( len <= 0 )

{

printf("some error happen when read\n");

event\_free(ev);

close(fd);

return ;

}

msg[len] = '\0';

printf("recv the client msg: %s", msg);

char reply\_msg[4096] = "I have recvieced the msg: ";

strcat(reply\_msg + strlen(reply\_msg), msg);

write(fd, reply\_msg, strlen(reply\_msg) );

}

typedef struct sockaddr SA;

int tcp\_server\_init(int port, int listen\_num)

{

int errno\_save;

evutil\_socket\_t listener;

listener = ::socket(AF\_INET, SOCK\_STREAM, 0);

if( listener == -1 )

return -1;

//允许多次绑定同一个地址。要用在socket和bind之间

evutil\_make\_listen\_socket\_reuseable(listener);

struct sockaddr\_in sin;

sin.sin\_family = AF\_INET;

sin.sin\_addr.s\_addr = 0;

sin.sin\_port = htons(port);

if( ::bind(listener, (SA\*)&sin, sizeof(sin)) < 0 )

goto error;

if( ::listen(listener, listen\_num) < 0)

goto error;

//跨平台统一接口，将套接字设置为非阻塞状态

evutil\_make\_socket\_nonblocking(listener);

return listener;

error:

errno\_save = errno;

evutil\_closesocket(listener);

errno = errno\_save;

return -1;

}

[复制代码](javascript:void(0);)

# 中等：

## 客户端代码：

[复制代码](javascript:void(0);)

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<errno.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<event.h>

#include<event2/bufferevent.h>

#include<event2/buffer.h>

#include<event2/util.h>

int tcp\_connect\_server(const char\* server\_ip, int port);

void cmd\_msg\_cb(int fd, short events, void\* arg);

void server\_msg\_cb(struct bufferevent\* bev, void\* arg);

void event\_cb(struct bufferevent \*bev, short event, void \*arg);

int main(int argc, char\*\* argv)

{

if( argc < 3 )

{

printf("please input 2 parameter\n");

return -1;

}

//两个参数依次是服务器端的IP地址、端口号

int sockfd = tcp\_connect\_server(argv[1], atoi(argv[2]));

if( sockfd == -1)

{

perror("tcp\_connect error ");

return -1;

}

printf("connect to server successful\n");

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

struct bufferevent\* bev = bufferevent\_socket\_new(base, sockfd,

BEV\_OPT\_CLOSE\_ON\_FREE);

//监听终端输入事件

struct event\* ev\_cmd = event\_new(base, STDIN\_FILENO,

EV\_READ | EV\_PERSIST, cmd\_msg\_cb,

(void\*)bev);

event\_add(ev\_cmd, NULL);

//当socket关闭时会用到回调参数

bufferevent\_setcb(bev, server\_msg\_cb, NULL, event\_cb, (void\*)ev\_cmd);

bufferevent\_enable(bev, EV\_READ | EV\_PERSIST);

event\_base\_dispatch(base);

printf("finished \n");

return 0;

}

void cmd\_msg\_cb(int fd, short events, void\* arg)

{

char msg[1024];

int ret = read(fd, msg, sizeof(msg));

if( ret < 0 )

{

perror("read fail ");

exit(1);

}

struct bufferevent\* bev = (struct bufferevent\*)arg;

//把终端的消息发送给服务器端

bufferevent\_write(bev, msg, ret);

}

void server\_msg\_cb(struct bufferevent\* bev, void\* arg)

{

char msg[1024];

size\_t len = bufferevent\_read(bev, msg, sizeof(msg));

msg[len] = '\0';

printf("recv %s from server\n", msg);

}

void event\_cb(struct bufferevent \*bev, short event, void \*arg)

{

if (event & BEV\_EVENT\_EOF)

printf("connection closed\n");

else if (event & BEV\_EVENT\_ERROR)

printf("some other error\n");

//这将自动close套接字和free读写缓冲区

bufferevent\_free(bev);

struct event \*ev = (struct event\*)arg;

//因为socket已经没有，所以这个event也没有存在的必要了

event\_free(ev);

}

typedef struct sockaddr SA;

int tcp\_connect\_server(const char\* server\_ip, int port)

{

int sockfd, status, save\_errno;

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr) );

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(port);

status = inet\_aton(server\_ip, &server\_addr.sin\_addr);

if( status == 0 ) //the server\_ip is not valid value

{

errno = EINVAL;

return -1;

}

sockfd = ::socket(PF\_INET, SOCK\_STREAM, 0);

if( sockfd == -1 )

return sockfd;

status = ::connect(sockfd, (SA\*)&server\_addr, sizeof(server\_addr) );

if( status == -1 )

{

save\_errno = errno;

::close(sockfd);

errno = save\_errno; //the close may be error

return -1;

}

evutil\_make\_socket\_nonblocking(sockfd);

return sockfd;

}

[复制代码](javascript:void(0);)

## 服务器端代码：

[复制代码](javascript:void(0);)

#include<stdio.h>

#include<string.h>

#include<errno.h>

#include<event.h>

#include<event2/bufferevent.h>

void accept\_cb(int fd, short events, void\* arg);

void socket\_read\_cb(bufferevent\* bev, void\* arg);

void event\_cb(struct bufferevent \*bev, short event, void \*arg);

int tcp\_server\_init(int port, int listen\_num);

int main(int argc, char\*\* argv)

{

int listener = tcp\_server\_init(9999, 10);

if( listener == -1 )

{

perror(" tcp\_server\_init error ");

return -1;

}

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

//添加监听客户端请求连接事件

struct event\* ev\_listen = event\_new(base, listener, EV\_READ | EV\_PERSIST,

accept\_cb, base);

event\_add(ev\_listen, NULL);

event\_base\_dispatch(base);

event\_base\_free(base);

return 0;

}

void accept\_cb(int fd, short events, void\* arg)

{

evutil\_socket\_t sockfd;

struct sockaddr\_in client;

socklen\_t len = sizeof(client);

sockfd = ::accept(fd, (struct sockaddr\*)&client, &len );

evutil\_make\_socket\_nonblocking(sockfd);

printf("accept a client %d\n", sockfd);

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

bufferevent\* bev = bufferevent\_socket\_new(base, sockfd, BEV\_OPT\_CLOSE\_ON\_FREE);

bufferevent\_setcb(bev, socket\_read\_cb, NULL, event\_cb, arg);

bufferevent\_enable(bev, EV\_READ | EV\_PERSIST);

}

void socket\_read\_cb(bufferevent\* bev, void\* arg)

{

char msg[4096];

size\_t len = bufferevent\_read(bev, msg, sizeof(msg));

msg[len] = '\0';

printf("recv the client msg: %s", msg);

char reply\_msg[4096] = "I have recvieced the msg: ";

strcat(reply\_msg + strlen(reply\_msg), msg);

bufferevent\_write(bev, reply\_msg, strlen(reply\_msg));

}

void event\_cb(struct bufferevent \*bev, short event, void \*arg)

{

if (event & BEV\_EVENT\_EOF)

printf("connection closed\n");

else if (event & BEV\_EVENT\_ERROR)

printf("some other error\n");

//这将自动close套接字和free读写缓冲区

bufferevent\_free(bev);

}

typedef struct sockaddr SA;

int tcp\_server\_init(int port, int listen\_num)

{

int errno\_save;

evutil\_socket\_t listener;

listener = ::socket(AF\_INET, SOCK\_STREAM, 0);

if( listener == -1 )

return -1;

//允许多次绑定同一个地址。要用在socket和bind之间

evutil\_make\_listen\_socket\_reuseable(listener);

struct sockaddr\_in sin;

sin.sin\_family = AF\_INET;

sin.sin\_addr.s\_addr = 0;

sin.sin\_port = htons(port);

if( ::bind(listener, (SA\*)&sin, sizeof(sin)) < 0 )

goto error;

if( ::listen(listener, listen\_num) < 0)

goto error;

//跨平台统一接口，将套接字设置为非阻塞状态

evutil\_make\_socket\_nonblocking(listener);

return listener;

error:

errno\_save = errno;

evutil\_closesocket(listener);

errno = errno\_save;

return -1;

}

[复制代码](javascript:void(0);)

# 高等：

## 客户端代码：

[复制代码](javascript:void(0);)

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<errno.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<event.h>

#include<event2/bufferevent.h>

#include<event2/buffer.h>

#include<event2/util.h>

int tcp\_connect\_server(const char\* server\_ip, int port);

void cmd\_msg\_cb(int fd, short events, void\* arg);

void server\_msg\_cb(struct bufferevent\* bev, void\* arg);

void event\_cb(struct bufferevent \*bev, short event, void \*arg);

int main(int argc, char\*\* argv)

{

if( argc < 3 )

{

//两个参数依次是服务器端的IP地址、端口号

printf("please input 2 parameter\n");

return -1;

}

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

struct bufferevent\* bev = bufferevent\_socket\_new(base, -1,

BEV\_OPT\_CLOSE\_ON\_FREE);

//监听终端输入事件

struct event\* ev\_cmd = event\_new(base, STDIN\_FILENO,

EV\_READ | EV\_PERSIST,

cmd\_msg\_cb, (void\*)bev);

event\_add(ev\_cmd, NULL);

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr) );

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(atoi(argv[2]));

inet\_aton(argv[1], &server\_addr.sin\_addr);

bufferevent\_socket\_connect(bev, (struct sockaddr \*)&server\_addr,

sizeof(server\_addr));

bufferevent\_setcb(bev, server\_msg\_cb, NULL, event\_cb, (void\*)ev\_cmd);

bufferevent\_enable(bev, EV\_READ | EV\_PERSIST);

event\_base\_dispatch(base);

printf("finished \n");

return 0;

}

void cmd\_msg\_cb(int fd, short events, void\* arg)

{

char msg[1024];

int ret = read(fd, msg, sizeof(msg));

if( ret < 0 )

{

perror("read fail ");

exit(1);

}

struct bufferevent\* bev = (struct bufferevent\*)arg;

//把终端的消息发送给服务器端

bufferevent\_write(bev, msg, ret);

}

void server\_msg\_cb(struct bufferevent\* bev, void\* arg)

{

char msg[1024];

size\_t len = bufferevent\_read(bev, msg, sizeof(msg));

msg[len] = '\0';

printf("recv %s from server\n", msg);

}

void event\_cb(struct bufferevent \*bev, short event, void \*arg)

{

if (event & BEV\_EVENT\_EOF)

printf("connection closed\n");

else if (event & BEV\_EVENT\_ERROR)

printf("some other error\n");

else if( event & BEV\_EVENT\_CONNECTED)

{

printf("the client has connected to server\n");

return ;

}

//这将自动close套接字和free读写缓冲区

bufferevent\_free(bev);

struct event \*ev = (struct event\*)arg;

event\_free(ev);

}

[复制代码](javascript:void(0);)

## 服务器端代码：

[复制代码](javascript:void(0);)

#include<netinet/in.h>

#include<sys/socket.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<event.h>

#include<listener.h>

#include<bufferevent.h>

#include<thread.h>

void listener\_cb(evconnlistener \*listener, evutil\_socket\_t fd,

struct sockaddr \*sock, int socklen, void \*arg);

void socket\_read\_cb(bufferevent \*bev, void \*arg);

void socket\_event\_cb(bufferevent \*bev, short events, void \*arg);

int main()

{

//evthread\_use\_pthreads();//enable threads

struct sockaddr\_in sin;

memset(&sin, 0, sizeof(struct sockaddr\_in));

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(9999);

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

evconnlistener \*listener

= evconnlistener\_new\_bind(base, listener\_cb, base,

LEV\_OPT\_REUSEABLE|LEV\_OPT\_CLOSE\_ON\_FREE,

10, (struct sockaddr\*)&sin,

sizeof(struct sockaddr\_in));

event\_base\_dispatch(base);

evconnlistener\_free(listener);

event\_base\_free(base);

return 0;

}

//一个新客户端连接上服务器了

//当此函数被调用时，libevent已经帮我们accept了这个客户端。该客户端的

//文件描述符为fd

void listener\_cb(evconnlistener \*listener, evutil\_socket\_t fd,

struct sockaddr \*sock, int socklen, void \*arg)

{

printf("accept a client %d\n", fd);

event\_base \*base = (event\_base\*)arg;

//为这个客户端分配一个bufferevent

bufferevent \*bev = bufferevent\_socket\_new(base, fd,

BEV\_OPT\_CLOSE\_ON\_FREE);

bufferevent\_setcb(bev, socket\_read\_cb, NULL, socket\_event\_cb, NULL);

bufferevent\_enable(bev, EV\_READ | EV\_PERSIST);

}

void socket\_read\_cb(bufferevent \*bev, void \*arg)

{

char msg[4096];

size\_t len = bufferevent\_read(bev, msg, sizeof(msg)-1 );

msg[len] = '\0';

printf("server read the data %s\n", msg);

char reply[] = "I has read your data";

bufferevent\_write(bev, reply, strlen(reply) );

}

void socket\_event\_cb(bufferevent \*bev, short events, void \*arg)

{

if (events & BEV\_EVENT\_EOF)

printf("connection closed\n");

else if (events & BEV\_EVENT\_ERROR)

printf("some other error\n");

//这将自动close套接字和free读写缓冲区

bufferevent\_free(bev);

}

[复制代码](javascript:void(0);)

http://blog.csdn.net/luotuo44/article/details/39670221