Libevent笔记

Libevent简介

libevent是一个基于事件触发的网络库，适用于windows、linux、bsd等多种平台，内部使用select、epoll、kqueue等系统调用管理事件机制。

官网：http://libevent.org/

特点：

- 事件驱动，高性能;
- 轻量级，专注于网络，不如ACE那么臃肿庞大，只提供了简单的网络API的封装，线程池，内存池，递归锁等均需要自己实现;
- 开放源码，代码相当精炼、易读;
- 跨平台，支持Windows、Linux、BSD和Mac OS;
- 支持多种I/O多路复用技术（epoll、poll、dev/poll、select和kqueue等），在不同的操作系统下，做了多路复用模型的抽象，可以选择使用不同的模型，通过事件函数提供服务;
- 支持I/O，定时器和信号等事件;
- 采用Reactor模式;

源码组织结构

Libevent的源代码虽然都在一层文件夹下面，但是其代码分类还是相当清晰的，主要可分为头文件、内部使用的头文件、辅助功能函数、日志、libevent框架、对系统I/O多路复用机制的封装、信号管理、定时事件管理、缓冲区管理、基本数据结构和基于libevent的两个实用库等几个部分，有些部分可能就是一个源文件。

源代码中的test部分就不在我们关注的范畴了。

1）头文件

主要就是event.h：事件宏定义、接口函数声明，主要结构体event的声明;

2）内部头文件

xxx-internal.h：内部数据结构和函数，对外不可见，以达到信息隐藏的目的;

3）libevent框架

event.c：event整体框架的代码实现;

4）对系统I/O多路复用机制的封装

epoll.c：对epoll的封装;

select.c：对select的封装;

devpoll.c：对dev/poll的封装;

kqueue.c：对kqueue的封装;

5）定时事件管理

min-heap.h：其实就是一个以时间作为key的小根堆结构；

6）信号管理

signal.c：对信号事件的处理；

7）辅助功能函数

evutil.h 和evutil.c：一些辅助功能函数，包括创建socket pair和一些时间操作函数：加、减和比较等。

8）日志

log.h和log.c：log日志函数

9）缓冲区管理

evbuffer.c和buffer.c：libevent对缓冲区的封装；

10）基本数据结构

compat/sys下的两个源文件：queue.h是libevent基本数据结构的实现，包括链表，双向链表，队列等；_libevent_time.h：一些用于时间操作的结构体定义、函数和宏定义；

11）实用网络库

http和evdns：是基于libevent实现的http服务器和异步dns查询库；

linux下源码安装

这里以libevent-2.0.19为例，最新版本详见官网（http://libevent.org/）。

wget https://github.com/downloads/libevent/libevent/libevent-2.0.19-stable.tar.gz

tar zxvf libevent-2.0.19-stable.tar.gz

cd libevent-2.0.19-stable

./configure && make && make install

报错：

error while loading shared libraries: libevent-2.0.so.5: cannot open shared object file: No such file or directory

解决办法：

On a 32 bit system:

ln -s /usr/local/lib/libevent-2.0.so.5 /usr/lib/libevent-2.0.so.5

On a 64 bit system:

ln -s /usr/local/lib/libevent-2.0.so.5 /usr/lib64/libevent-2.0.so.5

示例

1、获取版本

```
// gcc getVersion.c -o getVersion -levent

#include <event.h>
#include <stdio.h>

int main()
{
    const char *version = event_get_version();
    printf("%s\n",version);
    return 0;
}
```

2、一个简单的**http**服务器

```
// g++ libHttpServer.cpp -o libHttpServer -levent

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <sys/stat.h>
#include <event2/event.h>
#include <event2/http.h>
#include <event2/buffer.h>

#define LIBSRVR_SIGNATURE "libsrvr v 0.0.1"
#define LIBSRVR_HTDOCS "/"
#define LIBSRVR_INDEX "/index.html"

// Libsrvr http server and base struct
struct evhttp *libsrvr;
struct event_base *libbase;
// Libsrvr options
struct _options
{
   int port;
   char *address;
```

```
    int verbose;
} options;

void router (struct evhttp_request *r, void *arg)
{
    const char *uri = evhttp_request_get_uri (r);

    char *static_file =
        (char *) malloc (strlen (LIBSRVR_HTDOCS) + strlen (uri) +
                         strlen (LIBSRVR_INDEX) + 1);
    stpcpy (stpcpy (static_file, LIBSRVR_HTDOCS), uri);

    bool file_exists = true;
    struct stat st;
    if (stat (static_file, &st) == -1)
    {
        file_exists = false;
        evhttp_send_error (r, HTTP_NOTFOUND, "NOTFOUND");
    }
    else
    {
        if (S_ISDIR (st.st_mode))
        {
            strcat (static_file, LIBSRVR_INDEX);
            if (stat (static_file, &st) == -1)
            {
                file_exists = false;
                evhttp_send_error (r, HTTP_NOTFOUND, "NOTFOUND");
            }
        }
    }

    if (file_exists)
    {
        int file_size = st.st_size;
```

```
        char *html;
        html = (char *) alloca (file_size);

        if (file_size != 0)
        {
            FILE *fp = fopen (static_file, "r");
            fread (html, 1, file_size, fp);
            fclose (fp);
        }

        struct evbuffer *buffer;
        buffer = evbuffer_new ();

        struct evkeyvalq *headers = evhttp_request_get_output_headers (r);
        evhttp_add_header (headers, "Content-Type", "text/html; charset=UTF-8");
        evhttp_add_header (headers, "Server", LIBSRVR_SIGNATURE);

        evbuffer_add_printf (buffer, "%s", html);
        evhttp_send_reply (r, HTTP_OK, "OK", buffer);
        evbuffer_free (buffer);

        if (options.verbose)
            fprintf (stderr, "%s\t%d\n", static_file, file_size);
    }
    else
    {
        if (options.verbose)
            fprintf (stderr, "%s\t%s\n", static_file, "404 Not Found");
    }

    free (static_file);
}

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

```
{
   int opt;

   options.port = 4080;
   options.address = "0.0.0.0";
   options.verbose = 0;

   while ((opt = getopt (argc, argv, "p:vh")) != -1)
   {
      switch (opt)
      {
      case 'p':
         options.port = atoi (optarg);
         break;
      case 'v':
         options.verbose = 1;
         break;
      case 'h':
         printf ("Usage: ./libsrvr -p port -v[erbose] -h[elp]\n");
         exit (1);
      }
   }

   libbase = event_base_new ();
   libsrvr = evhttp_new (libbase);
   evhttp_bind_socket (libsrvr, options.address, options.port);
   evhttp_set_gencb (libsrvr, router, NULL);
   event_base_dispatch (libbase);

   return 0;
}
```

3、一个简单的**echoServer**

服务端代码:

```
// gcc echoServer.c -o echoServer -levent

#include <event2/listener.h>
#include <event2/bufferevent.h>
#include <event2/buffer.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>

static void echo_read_cb(struct bufferevent *bev, void *ctx)
{
   /* This callback is invoked when there is data to read on bev. */
   struct evbuffer *input = bufferevent_get_input(bev);
   struct evbuffer *output = bufferevent_get_output(bev);

   /* Copy all the data from the input buffer to the output buffer. */
   evbuffer_add_buffer(output, input);
}

static void echo_event_cb(struct bufferevent *bev, short events, void *ctx)
{
   if (events & BEV_EVENT_ERROR)
      perror("Error from bufferevent");
   if (events & (BEV_EVENT_EOF | BEV_EVENT_ERROR))
   {
      bufferevent_free(bev);
   }
}

static void accept_conn_cb(struct evconnlistener *listener,
                    evutil_socket_t fd, struct sockaddr *address, int socklen,
                    void *ctx)
{
```

```
    /* We got a new connection! Set up a bufferevent for it. */
    struct event_base *base = evconnlistener_get_base(listener);
    struct bufferevent *bev = bufferevent_socket_new(
                                base, fd, BEV_OPT_CLOSE_ON_FREE);

    bufferevent_setcb(bev, echo_read_cb, NULL, echo_event_cb, NULL);

    bufferevent_enable(bev, EV_READ|EV_WRITE);
}

static void accept_error_cb(struct evconnlistener *listener, void *ctx)
{
    struct event_base *base = evconnlistener_get_base(listener);
    int err = EVUTIL_SOCKET_ERROR();
    fprintf(stderr, "Got an error %d (%s) on the listener. "
            "Shutting down.\n", err, evutil_socket_error_to_string(err));

    event_base_loopexit(base, NULL);
}

int main(int argc, char **argv)
{
    struct event_base *base;
    struct evconnlistener *listener;
    struct sockaddr_in sin;

    int port = 50000;

    if (argc > 1)
    {
        port = atoi(argv[1]);
    }
    if (port<=0 || port>65535)
    {
        puts("Invalid port");
```

```
    return 1;
  }

  base = event_base_new();
  if (!base)
  {
    puts("Couldn't open event base");
    return 1;
  }

  /* Clear the sockaddr before using it, in case there are extra
   * platform-specific fields that can mess us up. */
  memset(&sin, 0, sizeof(sin));
  /* This is an INET address */
  sin.sin_family = AF_INET;
  /* Listen on 0.0.0.0 */
  sin.sin_addr.s_addr = htonl(0);
  /* Listen on the given port. */
  sin.sin_port = htons(port);

  listener = evconnlistener_new_bind(base, accept_conn_cb, NULL,
              LEV_OPT_CLOSE_ON_FREE|LEV_OPT_REUSEABLE, -1,
              (struct sockaddr*)&sin, sizeof(sin));

  if (!listener)
  {
    perror("Couldn't create listener");
    return 1;
  }
  evconnlistener_set_error_cb(listener, accept_error_cb);
  event_base_dispatch(base);

  return 0;
}
```

客户端代码（python）：

```
#! /usr/bin/python

import socket
sock=socket.socket(socket.AF_INET,socket.SOCK_STREAM,0)
sock.connect(('127.0.0.1',50000))
sock.send(raw_input("Input : "))
print "received : ",sock.recv(1024)
sock.close()
```

4、一个timer程序

```
// gcc timer.c -o timer -levent
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <event2/event.h>
#include <event2/event_struct.h>

#define N 300
#define BUFLEN 256

struct timeval lasttime;
struct ST_EventWithDescription
{
   struct event *p_event;
   int time_interval;
   char lable[BUFLEN];
};

static void timeout_cb(evutil_socket_t fd, short event, void *arg)
{
   struct timeval newtime, difference;
   struct ST_EventWithDescription *pSTEvent = arg;
   struct event *timeout = pSTEvent->p_event;
   double elapsed;

   evutil_gettimeofday(&newtime, NULL);
   evutil_timersub(&newtime, &lasttime, &difference);
   elapsed = difference.tv_sec + (difference.tv_usec / 1.0e6);

   printf("%s called at %d: %.3f seconds since my last work.\n",
        (char*)pSTEvent->lable,(int)newtime.tv_sec, elapsed);
   lasttime = newtime;
```

```
    struct timeval tv;
    evutil_timerclear(&tv);
    tv.tv_sec = pSTEvent->time_interval;
    event_add(timeout, &tv);
}

void setParam(struct ST_EventWithDescription *stEventDescription,
            struct event *m_event,int time_interval,char* m_lable)
{
    stEventDescription->p_event = m_event;
    stEventDescription->time_interval = time_interval;
    memset(stEventDescription->lable,0,sizeof(stEventDescription->lable));
    memcpy(stEventDescription->lable,m_lable,strlen(m_lable)+1);
}

void setTimeIntervalArr(int *arr,int n)
{
    int i;
    srand(time(NULL));
    for(i=0; i<n; ++i)
    {
        *(arr+i) = rand()%n + 1;
        //*(arr+i) = i+1;
    }
}

int main(int argc, char **argv)
{
    struct event timeout[N];
    struct ST_EventWithDescription stEvent[N];
    int time_interval[N];
    int i=0;

    struct timeval tv;
    struct event_base *base;
    int flags = 0;

    setTimeIntervalArr(time_interval,N);

    base = event_base_new();
    evutil_timerclear(&tv);

    for(i=0; i<N; ++i)
    {
        char buf[BUFLEN]= {0};
        sprintf(buf,"task%d",i+1);
        setParam(stEvent+i,timeout+i,time_interval[i],buf);
        event_assign(timeout+i, base, -1, flags, timeout_cb, (void*)(stEvent+i));
        event_add(timeout+i, &tv);
    }
```

```
    evutil_gettimeofday(&lasttime, NULL);
        event_base_dispatch(base);

    return (0);
}
```

完整地址：https://github.com/mike-zhang/testCodes/tree/master/libeventTest