bufferevent 与 socket - 穆穆兔兔

穆穆兔兔关注 - 1 粉丝 - 95 +加关注

http://blog.sina.com.cn/s/blog_56dee71a0100qx4s.html

很多时候，除了响应事件之外，应用还希望做一定的数据缓冲。比如说，写入数据的时候，通常的运行模式是：

l 决定要向连接写入一些数据，把数据放入到缓冲区中

l 等待连接可以写入

l 写入尽量多的数据

l 记住写入了多少数据，如果还有更多数据要写入，等待连接再次可以写入

1

这种缓冲IO模式很通用，libevent为此提供了一种通用机制，即bufferevent。bufferevent由一个底层的传输端口（如套接字），一个读取缓冲区和一个写入缓冲区组成。与通常的事件在底层传输端口已经就绪，可以读取或者写入的时候执行回调不同的是，bufferevent在读取或者写入了足够量的数据之后调用用户提供的回调。

bufferevent 的简单范例

这里选取了 Libevent 的一个范例程序 hello-world.c 来看看 Libevent 的用法：

```
#include <string.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#ifndef WIN32
#include <netinet/in.h>
# ifdef _XOPEN_SOURCE_EXTENDED
#  include <arpa/inet.h>
# endif
#include <sys/socket.h>
#endif
```


2

研究 **bufferevent** 的关键代码

这里只研究基于 socket 的 bufferevent。从上面 bufferevent 的使用可以看出，有几个关键函数：

1. 开始需要调用 bufferevent_socket_new 创建一个 bufferevent
2. 调用 bufferevent_setcb 设置回调函数
3. 调用 bufferevent_write 写入数据
4. 调用 bufferevent_free 释放 bufferevent

bufferevent_socket_new 的源码以及分析如下：

其中 bufferevent_init_common 函数实现为：

```
int
bufferevent_init_common(struct bufferevent_private *bufev_private,
    struct event_base *base,
    const struct bufferevent_ops *ops,
    enum bufferevent_options options)
{
  struct bufferevent *bufev = &bufev_private->bev;
```


3

bufferevent 创建成功之后，fd 上存在数据可读则调用 bufferevent_readcb

这里比较关键的函数为 evbuffer_read：

```
int
evbuffer_read(struct evbuffer *buf, evutil_socket_t fd, int howmuch)
{
  struct evbuffer_chain **chainp;
  int n;
  int result;

#ifdef USE_IOVEC_IMPL
  int nvecs, i, remaining;
#else
  struct evbuffer_chain *chain;
  unsigned char *p;
#endif

  EVBUFFER_LOCK(buf);
```


4

读完了 bufferevent_readcb 接下来再看看 bufferevent_writecb：

bufferevent_writecb 中比较关键的一个函数为 evbuffer_write_atmost：

```
int
evbuffer_write_atmost(struct evbuffer *buffer, evutil_socket_t fd,
    ev_ssize_t howmuch)
{
  int n = -1;

  EVBUFFER_LOCK(buffer);
```

代码读到这里，对于 bufferevent 的创建、socket 读写已经有了一定的了解，下面再看看 bufferevent_write，此函数实际上只是直接向输出缓冲区写入数据，缓冲区写入数据后，会调用回调 bufferevent_socket_outbuf_cb（创建 bufferevent 时设置的），此回调工作内容比较简单，主要就是将 ev_write 注册到 base 中去：

```
static void
bufferevent_socket_outbuf_cb(struct evbuffer *buf,
```


5

```
    const struct evbuffer_cb_info *cbinfo,
    void *arg)
{
  struct bufferevent *bufev = arg;
  struct bufferevent_private *bufev_p =
      EVUTIL_UPCAST(bufev, struct bufferevent_private, bev);
```

最后来看看释放过程：

```
void
bufferevent_free(struct bufferevent *bufev)
{
  BEV_LOCK(bufev);
```

更多详细的内容还需要更进一步阅读源码。

6