网络服务器的开发，主要在linux上，底层都是使用epoll进行异步IO处理。而开发者最常见的工作环境确是mac，不支持epoll，但有一个类似epoll的kqueue。为了方便开发与调试，我把自己的[https://github.com/yedf/handy](http://link.zhihu.com/?target=http%3A//blog.csdn.net/dongfuye/article/details/github.com/yedf/handy" \t "_blank)库移植到了mac平台上，把epoll的调用改成kqueue。

移植过程中，网上居然没有搜到kqueue的使用例子，让我惊讶不已。为了让大家不用像我一样再次花费大力气搞定kqueue，我整理了一个简单清晰可运行的kqueue例子，供大家参考。

kqueue一共有几个函数：

//类似epoll\_create

int kqueue(void);

//组合了epoll\_ctl及epoll\_wait功能，changelist与nchanges为修改列表，eventlist与nevents为返回的事件列表

int kevent(int kq, const struct kevent \*changelist, int nchanges, struct kevent \*eventlist, int nevents, const struct timespec \*timeout);

//设定kevent参数的宏，详细解释见后面的调用示例

EV\_SET(&kev, ident, filter, flags, fflags, data, udata);

下面是调用示例，仅为代码片段，解释一下大致的工作过程

*//创建kqueue，与epoll类似*

**int** epollfd **=** kqueue();

*//添加或者修改fd，类似epoll\_ctl，但kqueue的read/write两个事件是分开的*

**struct** kevent ev[2];

**int** n **=** 0;

**if** (events **&** kReadEvent) {

*//添加Read事件，EVFILT\_READ表示READ事件，操作为添加或者打开，多次重复操作没有副作用*

EV\_SET(**&**ev[n**++**], fd, EVFILT\_READ, EV\_ADD**|**EV\_ENABLE, 0, 0, (**void\***)(**intptr\_t**)fd);

} **else** **if** (modify){

*//删除Read事件，EVFILT\_READ表示READ事件，操作为添加或者打开，多次重复操作没有副作用*

EV\_SET(**&**ev[n**++**], fd, EVFILT\_READ, EV\_DELETE, 0, 0, (**void\***)(**intptr\_t**)fd);

}

*//更新Write相关事件*

**if** (events **&** kWriteEvent) {

EV\_SET(**&**ev[n**++**], fd, EVFILT\_WRITE, EV\_ADD**|**EV\_ENABLE, 0, 0, (**void\***)(**intptr\_t**)fd);

} **else** **if** (modify){

EV\_SET(**&**ev[n**++**], fd, EVFILT\_WRITE, EV\_DELETE, 0, 0, (**void\***)(**intptr\_t**)fd);

}

printf("%s fd %d events read %d write %d\n",

modify **?** "mod" **:** "add", fd, events **&** kReadEvent, events **&** kWriteEvent);

*//调用kevent，应用更改*

**int** r **=** kevent(efd, ev, n, NULL, 0, NULL);

*//获取ready的fd，类似epoll\_wait*

**struct** timespec timeout;

timeout.tv\_sec **=** waitms **/** 1000;

timeout.tv\_nsec **=** (waitms **%** 1000) **\*** 1000 **\*** 1000;

**const** **int** kMaxEvents **=** 20;

**struct** kevent activeEvs[kMaxEvents];

**int** n **=** kevent(efd, NULL, 0, activeEvs, kMaxEvents, **&**timeout);

*//处理IO事件*

**for** (**int** i **=** 0; i **<** n; i **++**) {

**int** fd **=** (**int**)(**intptr\_t**)activeEvs[i].udata;

**int** events **=** activeEvs[i].filter;

**if** (events **==** EVFILT\_READ) {

handleRead(efd, fd);

} **else** **if** (events **==** EVFILT\_WRITE) {

handleWrite(efd, fd);

}

}

注意kevent与epoll最大的不同在于READ/WRITE事件是分开注册并且分开返回的，而Epoll则是一个fd一次返回读和写事件，用标志位来判断。   
可以运行的代码如下：[kqueue-example](http://link.zhihu.com/?target=https%3A//github.com/yedf/kqueue-example" \t "_blank)（[https://github.com/yedf/handy](http://link.zhihu.com/?target=https%3A//github.com/yedf/handy)对kqueue提供了封装版本）

另外例子代码并非是kqueue的最佳用法，只是方便把epoll的代码移植到mac上。

如果要最大限度的利用kqueue的性能，最好合并多个READ/WRITE事件的更新，