Comparing Two High-Performance I/O Design Patterns

<http://www.artima.com/articles/io_design_patternsP.html>

介绍了Unix世界和Windows世界下非阻塞异步操作的理念不同

同时可以参考：

Linux and I/O completion ports?: <http://stackoverflow.com/questions/2794535/linux-and-i-o-completion-ports>

If you're looking for something exactly like IOCP, you won't find it, because it doesn't exist.

Windows uses a notify on completion model (hence I/O *Completion* Ports). You start some operation asynchronously, and receive a notification when that operation has completed.

Linux applications (and most other Unix-alikes) generally use a notify on ready model. You receive a notification that the socket can be read from or written to without blocking. Then, you do the I/O operation, which will not block.

有关同步/异步，阻塞/非阻塞的区别和定义：

<http://stackoverflow.com/questions/2625493/asynchronous-vs-non-blocking>

在Windows的术语里，overlapped==asynchronous,而且在Windows的世界里并没有专门的non-blocking一说，除了在socket的select里。non-blcoking更多地出现在\*nix系统里。

“Asynchronous I/O in Windows for Unix Programmers” <http://tinyclouds.org/iocp-links.html>

epoll用法说明：

<http://www.cnblogs.com/carekee/articles/2760661.html>

<http://blog.chinaunix.net/uid-20665047-id-3701634.html>

主要思想就是，先只会创建一个maste/listenr套接字，然后将其加入epoll池，开始用epoll\_wait等待连接到来，此时到达的连接有可能是master套接字也可能是后来创建的slave套接字.如果是master，则调用accpet去创建slave套接字，因为此时accept必不会阻塞。如果是slave套接字，则调用具体的执行函数。

listen函数说明：

<http://cpp.ezbty.org/content/science_doc/%E7%BD%91%E7%BB%9C%E7%BC%96%E7%A8%8Bsocket%E4%B9%8Blisten%E5%87%BD%E6%95%B0>

accept函数说明：

<http://cpp.ezbty.org/content/science_doc/%E7%BD%91%E7%BB%9C%E7%BC%96%E7%A8%8Bsocket%E4%B9%8Baccept%E5%87%BD%E6%95%B0>

select， poll， 以及epoll的区别。

<http://wenku.baidu.com/view/0ea86ffdc8d376eeaeaa3198.html>

简单来说是逐渐进步的，select用位来描述FD\_SET，linux上最大限制为1024，poll前进一步，用了一个poolfd的指针数组结构存放FD\_SET，该参数从形式上优于select，即只会列出poll关心的套接字，而不是像select那样只是给出一个范围，所以当监听的套接字十分离散时，poll的效率较select要高。但其个数在linux上也是有2048的限制。而且对于select和poll，内核不管FD\_SET中的socket是否激活，每次查询都要扫描所有监视端口，所以执行复杂度对于等待socket个数都是是线性增长的。select和poll还有一个缺点是每次都要从用户态拷贝socket和fd信息，并挂到进程的等待队列上，由于用户进程使用的方式都是在一个while循环中反复调用select和poll函数，所以每次都要做重复的劳动，效率也低。

epoll克服了以上缺点，首先FD\_SET容量几乎无限制，其次执行效率高，对不激活端口不消耗CPU，所以对于等待端口不是全忙的情况，epoll有更高的执行效率。而且epoll在执行epoll\_create和epoll\_ctrl时已经把用户信息拷贝到内核态，epoll\_wait时不会做多余的拷贝动作。

select.poll以及epoll的内部实现机制调研

<http://wenku.baidu.com/view/4b9f4dd3360cba1aa811daf0.html>

一个改进版本，而且比较清晰：

<http://blog.csdn.net/nailding2/article/details/6858199>

一个非常详细的说明，select和poll在内核的实现机制，有图更易理解 <http://blog.chinaunix.net/uid-24789420-id-397440.html>

一个比较好的总结-如何在驱动中支持poll：<http://blog.chinaunix.net/uid-25873032-id-2240153.html>

一个讲解得不错的有关socket编程的文章[vc 网络编程（socket）TCP/UDP 介绍](http://blog.csdn.net/wwqingyue/article/details/8792608)：

<http://blog.csdn.net/wwqingyue/article/details/8792608>

等待队列：

Linux中等待队列机制分析 ： <http://bbs.ednchina.com/BLOG_ARTICLE_133368.HTM>

Linux内核中等待队列的几种用法： <http://www.php100.com/html/webkaifa/Linux/2012/1016/11266.html>

注意高级应用和基本应用之间的区别

Linux的线程也是可以用wait\_queue来等待的，参考<http://www.embeddedlinux.org.cn/EssentialLinuxDeviceDrivers/final/ch03lev1sec1.html>

AIO support on Linux <http://stackoverflow.com/questions/7956731/aio-support-on-linux>

How to use epoll? A complete example in C: <https://banu.com/blog/2/how-to-use-epoll-a-complete-example-in-c/>

linux AIO （异步IO） 那点事儿： <http://cnodejs.org/topic/4f16442ccae1f4aa270010a7>

Linux kernel AIO这个奇葩： <http://www.wzxue.com/linux-kernel-aio%E8%BF%99%E4%B8%AA%E5%A5%87%E8%91%A9/>