[**再谈select, iocp, epoll,kqueue及各种I/O复用机制**](http://blog.csdn.net/yongche_shi/article/details/36405791)

标签： [epoll](http://www.csdn.net/tag/epoll" \t "_blank)[select](http://www.csdn.net/tag/select)[iocp](http://www.csdn.net/tag/iocp)[并发](http://www.csdn.net/tag/%e5%b9%b6%e5%8f%91)

2014-07-02 12:39 1161人阅读 [评论](http://blog.csdn.net/yongche_shi/article/details/36405791#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/yongche_shi/article/details/36405791#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

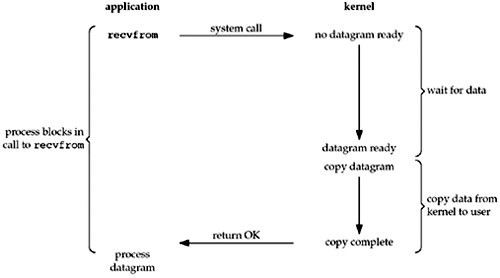
高并发（19） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

上篇文章太过幼稚，一派胡言，希望没有误导大家。。我也是菜鸟，学习本身就是一个不断追求真理的过程，希望能谅解：）首先感谢kasicass GG指出错误（已经指导我很多次了，非常感谢~~），然后我又查阅了一些资料，再次整理，more seriously and detailedly~

**首先，介绍几种常见的I/O模型及其区别**，如下：

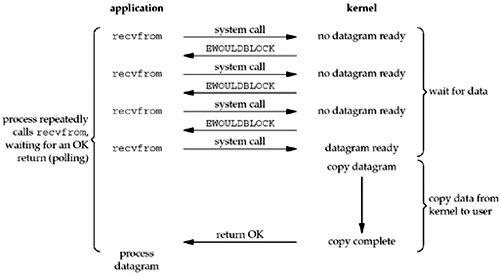
* blocking I/O
* nonblocking I/O
* I/O multiplexing (select and poll)
* signal driven I/O (SIGIO)
* asynchronous I/O (the POSIX aio\_functions)

**blocking I/O**  
这个不用多解释吧，阻塞套接字。下图是它调用过程的图示：



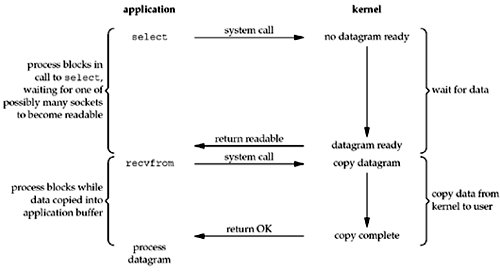
重点解释下上图，下面例子都会讲到。首先application调用 recvfrom()转入kernel，注意kernel有2个过程，wait for data和copy data from kernel to user。直到最后copy complete后，recvfrom()才返回。此过程一直是阻塞的。

**nonblocking I/O：**  
与blocking I/O对立的，非阻塞套接字，调用过程图如下：



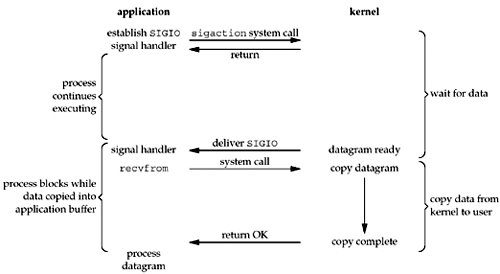
可以看见，如果直接操作它，那就是个轮询。。直到内核缓冲区有数据。

**I/O multiplexing (select and poll)**  
最常见的I/O复用模型，select。



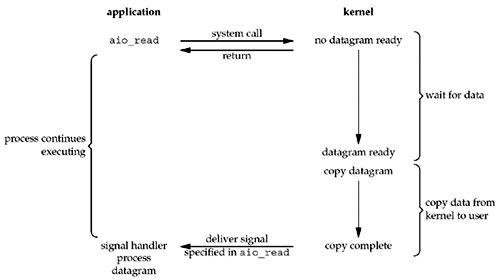
select先阻塞，有活动套接字才返回。与**blocking I/O**相比，select会有两次系统调用，但是select能处理多个套接字。

**signal driven I/O (SIGIO)**  
只有UNIX系统支持，感兴趣的课查阅相关资料



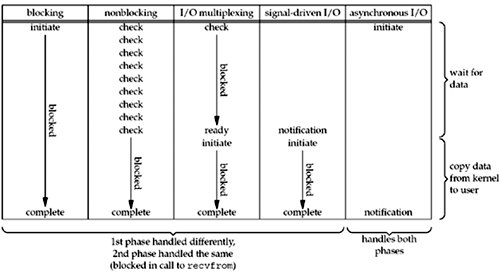
与**I/O multiplexing (select and poll)**相比，它的优势是，免去了select的阻塞与轮询，当有活跃套接字时，由注册的handler处理。

**asynchronous I/O (the POSIX aio\_functions)**  
很少有\*nix系统支持，windows的IOCP则是此模型



完全异步的I/O复用机制，因为纵观上面其它四种模型，至少都会在由kernel copy data to appliction时阻塞。而该模型是当copy完成后才通知application，可见是纯异步的。好像只有windows的完成端口是这个模型，效率也很出色。

**下面是以上五种模型的比较**

****

可以看出，越往后，阻塞越少，理论上效率也是最优。

=====================分割线==================================

5种模型的比较比较清晰了，剩下的就是把select,epoll,iocp,kqueue按号入座那就OK了。

select和iocp分别对应第3种与第5种模型，那么epoll与kqueue呢？其实也于select属于同一种模型，只是更高级一些，可以看作有了第4种模型的某些特性，如callback机制。

**那么，为什么epoll,kqueue比select高级？**

答案是，他们无**轮询**。因为他们用callback取代了。想想看，当套接字比较多的时候，每次select()都要通过遍历FD\_SETSIZE个Socket来完成调度,不管哪个Socket是活跃的,都遍历一遍。这会浪费很多CPU时间。如果能给套接字注册某个回调函数，当他们活跃时，自动完成相关操作，那就避免了轮询，这正是epoll与kqueue做的。

**windows or \*nix （IOCP or kqueue/epoll）？**

诚然，Windows的IOCP非常出色，目前很少有支持**asynchronous I/O**的系统，但是由于其系统本身的局限性，大型服务器还是在UNIX下。而且正如上面所述，kqueue/epoll 与 IOCP相比，就是多了一层从内核copy数据到应用层的阻塞，从而不能算作**asynchronous I/O类。**但是，这层小小的阻塞无足轻重，kqueue与epoll已经做得很优秀了。

**提供一致的接口，IO Design Patterns**

实际上，不管是哪种模型，都可以抽象一层出来，提供一致的接口，广为人知的有ACE,Libevent这些，他们都是跨平台的，而且他们自动选择最优的I/O复用机制，用户只需调用接口即可。说到这里又得说说2个设计模式，**Reactor**and **Proactor。**有一篇经典文章<http://www.artima.com/articles/io_design_patterns.html>值得阅读，Libevent是**Reactor**模型，ACE提供**Proactor**模型。实际都是对各种I/O复用机制的封装。

**Java nio包是什么I/O机制？**

我曾天真的认为java nio封装的是IOCP。。现在可以确定，目前的java本质是select()模型，可以检查/jre/bin/nio.dll得知。至于java服务器为什么效率还不错。。我也不得而知，可能是设计得比较好吧。。-\_-。

=====================分割线==================================

**总结一些重点：**

1. 只有IOCP是asynchronous I/O，其他机制或多或少都会有一点阻塞。
2. select低效是因为每次它都需要轮询。但低效也是相对的，视情况而定，也可通过良好的设计改善
3. epoll, kqueue是Reacor模式，IOCP是Proactor模式。
4. java nio包是select模型。。