【转】IO模型及select、poll、epoll和kqueue的区别

（一）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`)————异步IO模型最大的特点是 完成后发回通知。

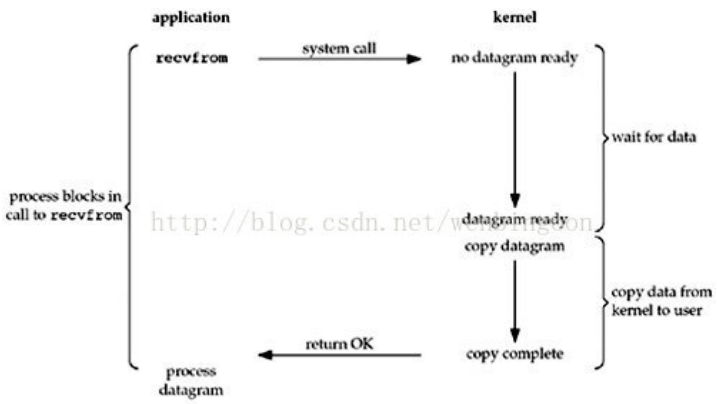
  阻塞与否，取决于实现IO交换的方式。

  异步阻塞是基于select，select函数本身的实现方式是阻塞的，而采用select函数有个好处就是它可以同时监听多个文件句柄.

  异步非阻塞直接在完成后通知，用户进程只需要发起一个IO操作然后立即返回，等IO操作真正的完成以后，应用程序会得到IO操作完成的通知，此时用户进程只需要对数据进行处理就好了，不需要进行实际的IO读写操作，因为真正的IO读取或者写入操作已经由内核完成了。

1 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。

2 nonblocking I/O：

与blocking I/O对立的，非阻塞套接字，调用过程图如下：

公告

昵称：超级大熊
园龄：2年6个月
粉丝：5
关注：1
+加关注

‹ 2018年12月 ›

日	一	二	三	四	五	六
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

搜索

常用链接

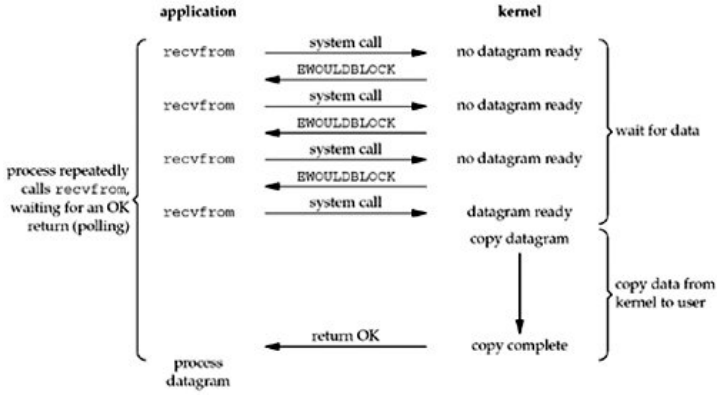
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

iOS(75)
ruby(9)
面试(9)
swift(7)
调试(5)
runtime(4)
性能优化(4)
音视频(2)
多线程(2)
runloop(2)
更多

随笔档案

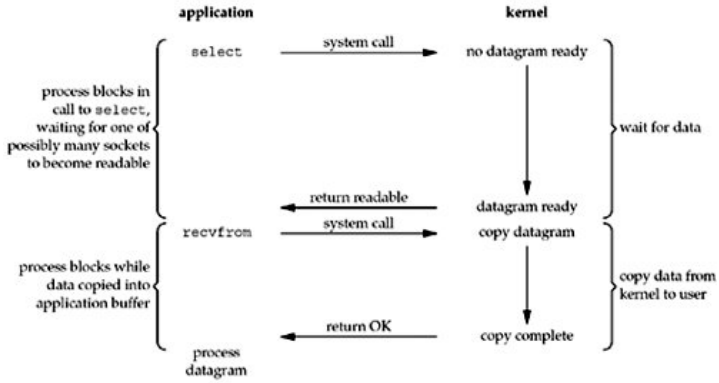
2018年12月 (1)
2018年9月 (4)
2018年7月 (3)
2018年6月 (3)
2018年5月 (8)
2018年4月 (2)
2018年1月 (3)
2017年12月 (2)



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。

3 I/O multiplexing (select and poll)

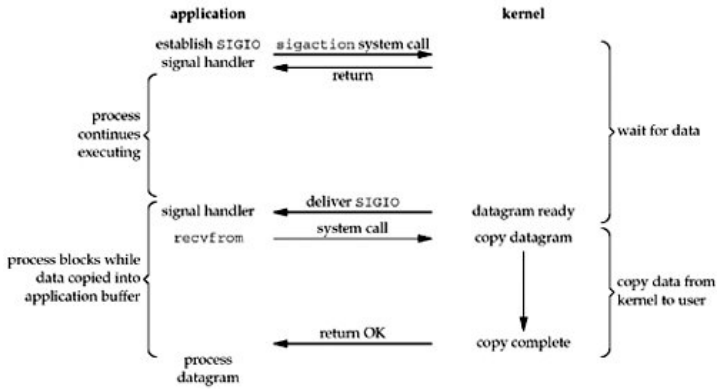
最常见的I/O复用模型，select。



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。**

4 signal driven I/O (SIGIO)

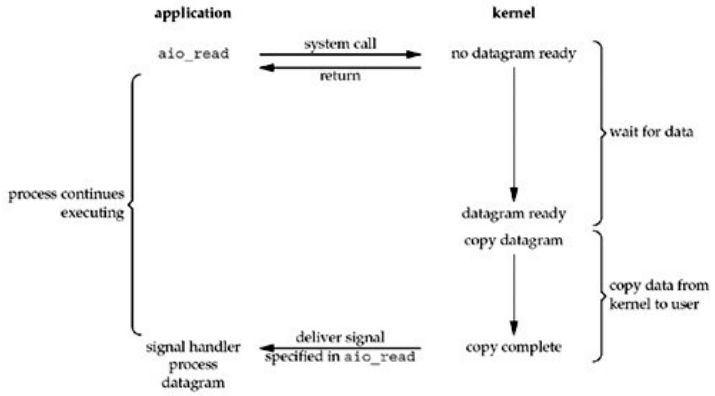
只有UNIX系统支持，感兴趣的课查阅相关资料



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。

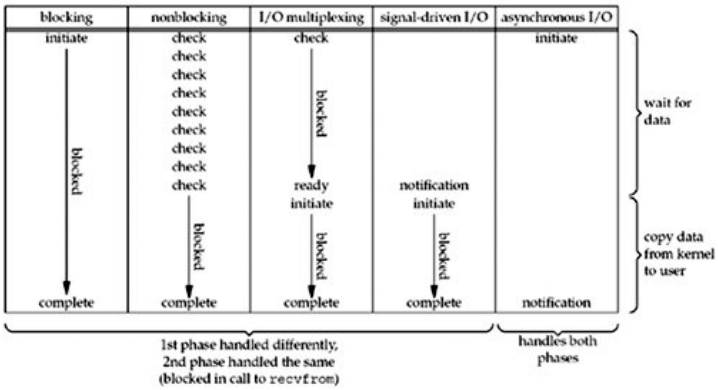
5 asynchronous I/O (the POSIX aio_functions)

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。

6 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。

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（基于reactor模式）**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**。见：**Reactor模式--VS--Proactor模式**。**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?

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、select是Reacor模式，IOCP是Proactor模式。**
4. **java nio包是select模型。。**

（二）epoll 与select的区别

1. 使用多进程或者多线程，但是这种方法会造成程序的复杂，而且对与进程与线程的创建维护也需要很多的开销。（Apache服务器是用的子进程的方式，优点可以隔离用户） （同步阻塞IO）

2.一种较好的方式为I/O多路转接（I/O multiplexing）（貌似也翻译多路复用），先构造一张有关描述符的列表（epoll中为队列），然后调用一个函数，直到这些描述符中的一个准备好时才返回，返回时告诉进程哪些I/O就绪。select和epoll这两个机制都是多路I/O机制的解决方案，select为POSIX标准中的，而epoll为Linux所特有的。

区别（epoll相对select优点）主要有三：

1.select的句柄数目受限，在linux/posix_types.h头文件有这样的声明：#define __FD_SETSIZE 1024 表示select最多同时监听1024个fd。而epoll没有，它的限制是最大的打开文件句柄数目。

2.epoll的最大好处是不会随着FD的数目增长而降低效率，在selec中采用轮询处理，其中的数据结构类似一个数组的数据结构，而epoll是维护一个队列，直接看队列是不是空就可以了。epoll只会对"活跃"的socket进行操作---这是因为在内核实现中epoll是根据每个fd上面的callback函数实现的。那么，只有"活跃"的socket才会主动的去调用 callback函数（把这个句柄加入队列），其他idle状态句柄则不会，在这点上，epoll实现了一个"伪"AIO。但是如果绝大部分的I/O都是“活跃的”，每个I/O端口使用率很高的话，epoll效率不一定比select高（可能是要维护队列复杂）。

3.使用mmap加速内核与用户空间的消息传递。无论是select,poll还是epoll都需要内核把FD消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll是通过内核于用户空间mmap同一块内存实现的。

关于epoll工作模式ET，LT

epoll有两种工作方式

ET：Edge Triggered，边缘触发。仅当状态发生变化时才会通知，epoll_wait返回。换句话，就是对于一个事件，只通知一次。且只支持非阻塞的socket。

LT：Level Triggered，电平触发（默认工作方式）。类似select/poll,只要还有没有处理的事件就会一直通知，以LT方式调用epoll接口的时候，它就相当于一个速度比较快的poll.支持阻塞和不阻塞的socket。

三 Linux并发网络编程模型

1 Apache 模型，简称 PPC （ Process Per Connection ， ）:为每个连接分配一个进程。主机分配给每个连接的时间和空间上代价较大，并且随着连接的增多，大量进程间切换开销也增长了。很难应对大量的客户并发连接。

2 TPC 模型（ Thread Per Connection ）：每个连接一个线程。和PCC类似。

3 select 模型：I/O多路复用技术。

.1 每个连接对应一个描述。select模型受限于 FD_SETSIZE即进程最大打开的描述符数linux2.6.35为1024，实际上linux每个进程所能打开描数字的个数仅受限于内存大小，然而在设计select的系统调用时，却是参考FD_SETSIZE的值。可通过重新编译内核更改此值，但不能根治此问题，对于百万级的用户连接请求 即便增加相应 进程数， 仍显得杯水车薪呀。

.2select每次都会扫描一个文件描述符的集合，这个集合的大小是作为select第一个参数传入的值。但是每个进程所能打开文件描述符若是增加了 ，扫描的效率也将减小。

.3内核到用户空间，采用内存复制传递文件描述上发生的信息。

4 poll 模型：I/O多路复用技术。poll模型将不会受限于FD_SETSIZE，因为内核所扫描的文件 描述符集合的大小是由用户指定的，即poll的第二个参数。但仍有扫描效率和内存拷贝问题。

5 pselect模型：I/O多路复用技术。同select。

6 epoll模型：

.1)无文件描述字大小限制仅与内存大小相关

.2)epoll返回时已经明确的知道哪个socket fd发生了什么事件，不用像select那样再一个个比对。

.3)内核到用户空间采用共享内存方式，传递消息。

四： FAQ

1、单个epoll并不能解决所有问题，特别是你的每个操作都比较费时的时候，因为epoll是串行处理的。 所以你有还是必要建立线程池来发挥更大的效能。

2、如果fd被注册到两个epoll中时，如果有时间发生则两个epoll都会触发事件。

3、如果注册到epoll中的fd被关闭，则其会自动被清除出epoll监听列表。
4、如果多个事件同时触发epoll，则多个事件会被联合在一起返回。
5、epoll_wait会一直监听epollhup事件发生，所以其不需要添加到events中。
6、为了避免大数据量io时，et模式下只处理一个fd,其他fd被饿死的情况发生。linux建议可以在fd联系到的结构中增加ready位，然后epoll_wait触发事件之后仅将其置位为ready模式，然后在下边轮询ready fd列表。

参考：

http://blog.csdn.net/ysu108/article/details/7570571

http://techbbs.zol.com.cn/1/8_2245.html

转自：http://blog.csdn.net/wenbingoon/article/details/9004512

标签: 服务器

超级大熊
关注 - 1
粉丝 - 5
+加关注

0 0

« 上一篇：【转】HTTP in iOS你看我就够
» 下一篇：iOS中几种定时器 - 控制了时间，就控制了一切

posted @ 2016-06-14 11:21 超级大熊 阅读(13925) 评论(1) 编辑 收藏

评论列表

#1楼 2018-04-09 23:14 wjc_小斯

博主, epoll 是有轮询的，没有callback这一说的

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部