100%弄明白5种IO模型

勤劳的小手
互联网/IT/java

从TCP发送数据的流程说起

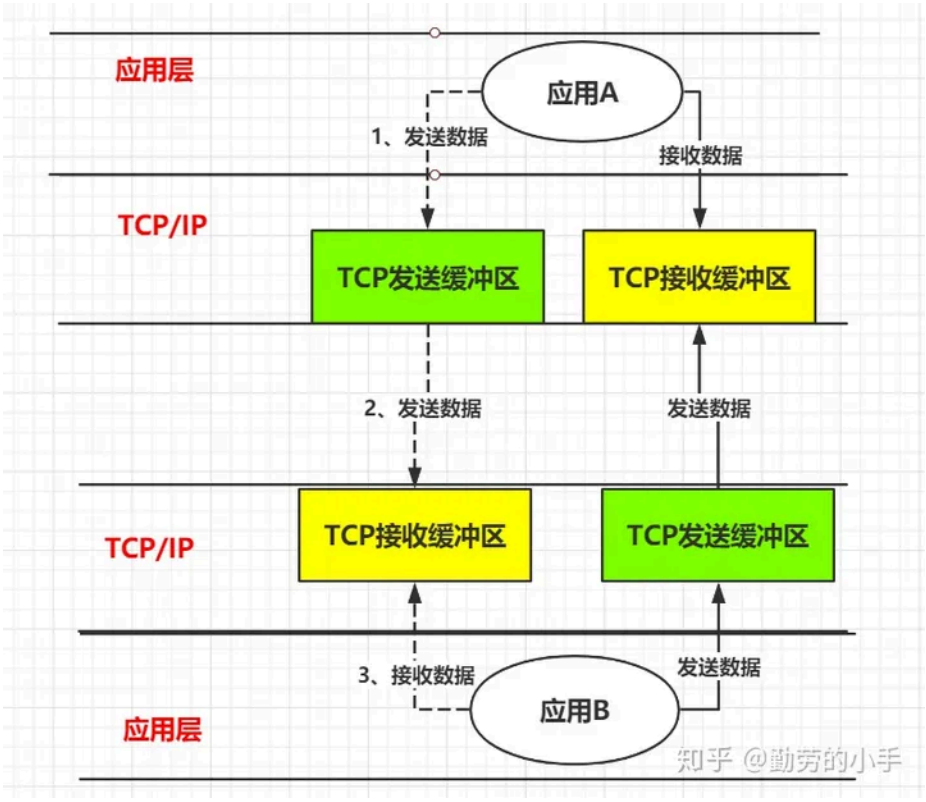
要深入的理解各种IO模型，那么必须先了解下产生各种IO的原因是什么，要知道这其中的本质问题那么我们就必须要知道一条消息是如何从一个人发送到另外一个人的；

以两个应用程序通讯为例，我们来了解一下当“A”向"B"发送一条消息，简单来说会经过如下流程：

第一步：应用A把消息发送到 TCP发送缓冲区。

第二步： TCP发送缓冲区再把消息发送出去，经过网络传递后，消息会发送到B服务器的TCP接收缓冲区。

第三步：B再从TCP接收缓冲区去读取属于自己的数据。



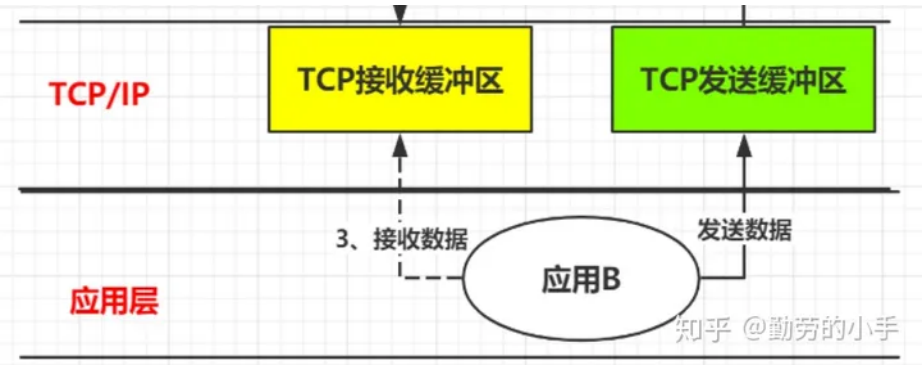
根据上图我们基本上了解消息发送要经过 应用A、应用A对应服务器的TCP发送缓冲区、经过网络传输后消息发送到了应用B对应服务器TCP接收缓冲区、然后最终B应用读取到消息。

如果理解了上面的消息发送流程，那么我们下面开始进入文章的主题；

知乎 首发于 必须深入理解的50个JAVA基础概念

阻塞IO |非阻塞IO

我们把视角切换到上面图中的第三步， 也就是应用B从TCP缓冲区中读取数据。



思考一个问题：

因为应用之间发送消息是间断性的，也就是说在上图中TCP缓冲区还没有接收到属于应用B该读取的消息时，那么此时应用B向TCP缓冲区发起读取申请，TCP接收缓冲区是应该马上告诉应用B 现在没有你的数据，还是说让应用B在这里等着，直到有数据再把数据交给应用B。

把这个问题应用到第一个步骤也是一样，应用A在向TCP发送缓冲区发送数据时，如果TCP发送缓冲区已经满了，那么是告诉应用A现在没空间了，还是让应用A等待着，等TCP发送缓冲区有空间了再把应用A的数据访拷贝到发送缓冲区。

什么是阻塞IO

如果上面的问题你已经思考过了，那么其实你已经明白了什么是阻塞IO了，所谓阻塞IO就是当应用B发起读取数据申请时，在内核数据没有准备好之前，应用B会一直处于等待数据状态，直到内核把数据准备好了交给应用B才结束。

术语描述：在应用调用recvfrom读取数据时，其系统调用直到数据包到达且被复制到应用缓冲区中或者发送错误时才返回，在此期间一直会等待，进程从调用到返回这段时间内都是被阻塞的称为阻塞IO；

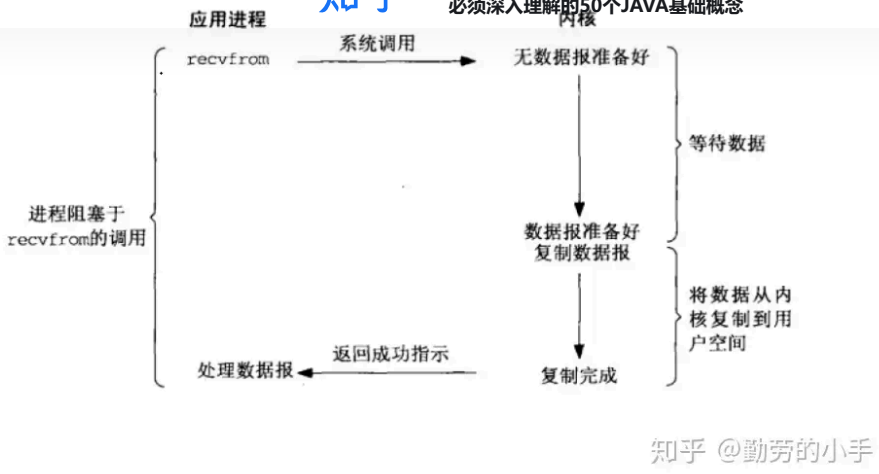
流程：

1、应用进程向内核发起recfrom读取数据。

2、准备数据报（应用进程阻塞）。

3、将数据从内核负责到应用空间。

4、复制完成后，返回成功提示。



什么是非阻塞IO

我敢保证如果你已经理解了阻塞IO，那么必定已经知道了什么是非阻塞IO。按照上面的思路，所谓非阻塞IO就是当应用B发起读取数据申请时，如果内核数据没有准备好会即刻告诉应用B，不会让B在这里等待。

术语：非阻塞IO是在应用调用recvfrom读取数据时，如果该缓冲区没有数据的话，就会直接返回一个EWOULDBLOCK错误，不会让应用一直等待中。在没有数据的时候会即刻返回错误标识，那也意味着如果应用要读取数据就需要不断的调用recvfrom请求，直到读取到它数据要的数据为止。

流程：

1、应用进程向内核发起recvfrom读取数据。

2、没有数据报准备好，即刻返回EWOULDBLOCK错误码。

3、应用进程向内核发起recvfrom读取数据。

4、已有数据包准备好就进行一下 步骤，否则还是返回错误码。

5、将数据从内核拷贝到用户空间。

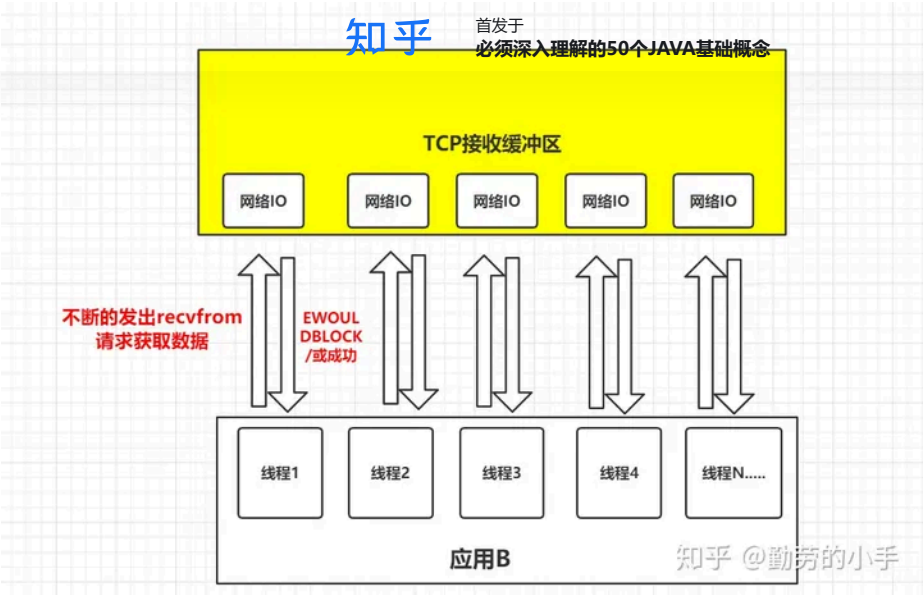
6、完成后，返回成功提示。

IO复用模型

如果你已经明白了非阻塞IO的工作模式，那么接下来我们继续了解IO复用模型的产生原因和思路。

思考一个问题：

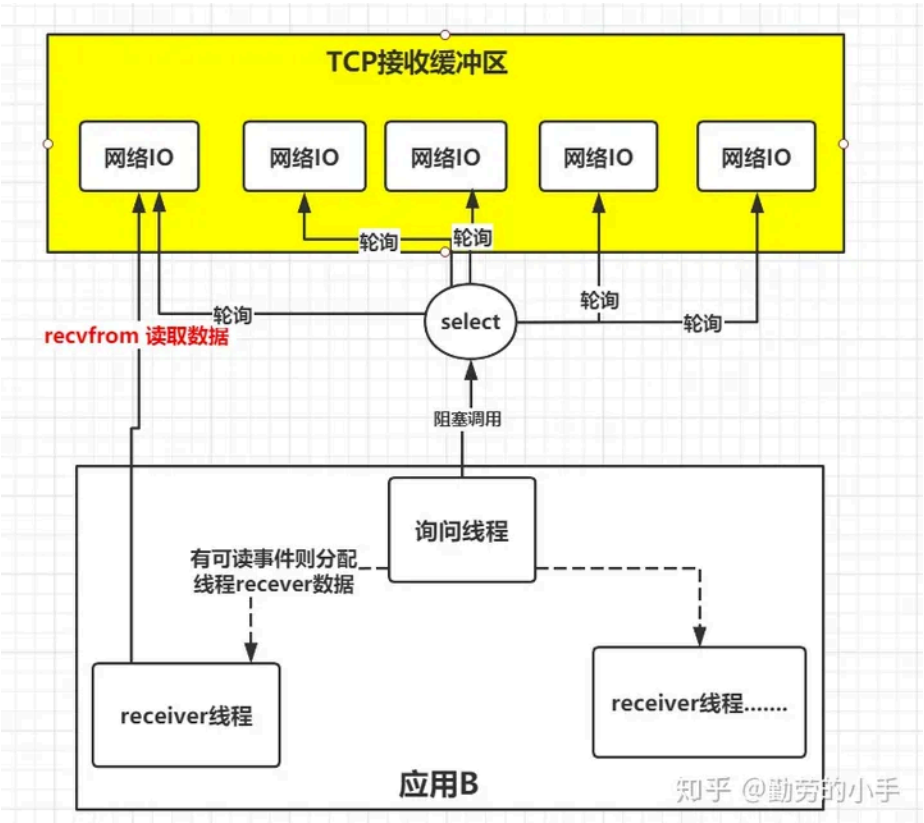
我们还是把视角放到应用B从TCP缓冲区中读取数据这个环节来。如果在并发的环境下，可能会N个人向应用B发送消息，这种情况下我们的应用就必须创建多个线程去读取数据，每个线程都会自己调用recvfrom 去读取数据。那么此时情况可能如下图：



如上图一样，并发情况下服务器很可能一瞬间会收到几十上百万的请求，这种情况下应用B就需要创建几十上百万的线程去读取数据，同时又因为应用线程是不知道什么时候会有数据读取，为了保证消息能及时读取到，那么这些线程自己必须不断的向内核发送recvfrom 请求来读取数据；

那么问题来了，这么多的线程不断调用recvfrom 请求数据，先不说服务器能不能扛得住这么多线程，就算扛得住那么很明显这种方式是不是太浪费资源了，线程是我们操作系统的宝贵资源，大量的线程用来去读取数据了，那么就意味着能做其它事情的线程就会少。

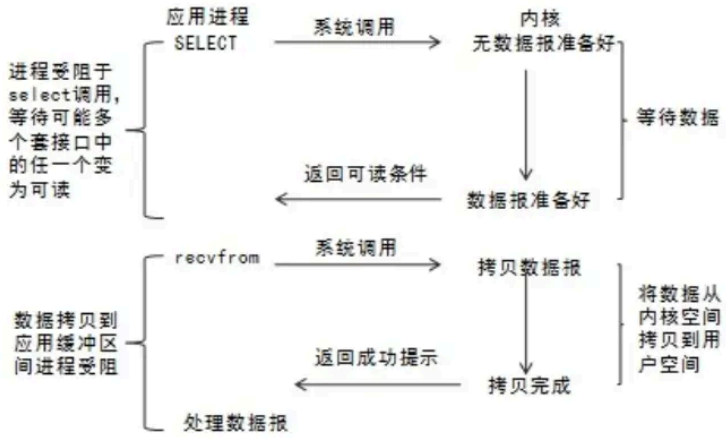
所以，有人就提出了一个思路，能不能提供一种方式，可以由一个线程监控多个网络请求（**我们后面将称为fd文件描述符，linux系统把所有网络请求以一个fd来标识**），这样就可以只需要一个或几个线程就可以完成数据状态询问的操作，当有数据准备就绪之后再分配对应的线程去读取数据，这么做就可以节省出大量的线程资源出来，这个就是IO复用模型的思路。



正如上图，IO复用模型的思路就是系统提供了一种函数可以同时监控多个fd的操作，这个函数就是我们常说到的select、poll、epoll函数，有了这个函数后，应用线程通过调用select函数就可以同

时监控多个fd，select函数监控的fd中只要有任何一个数据状态准备就绪了，select函数就会返回可读状态，这时询问线程再去通知处理数据的线程，对应线程此时再发起recvfrom请求去读取数据。

术语描述：进程通过将一个或多个fd传递给select，阻塞在select操作上，select帮我们侦测多个fd是否准备就绪，当有fd准备就绪时，select返回数据可读状态，应用程序再调用recvfrom读取数据。



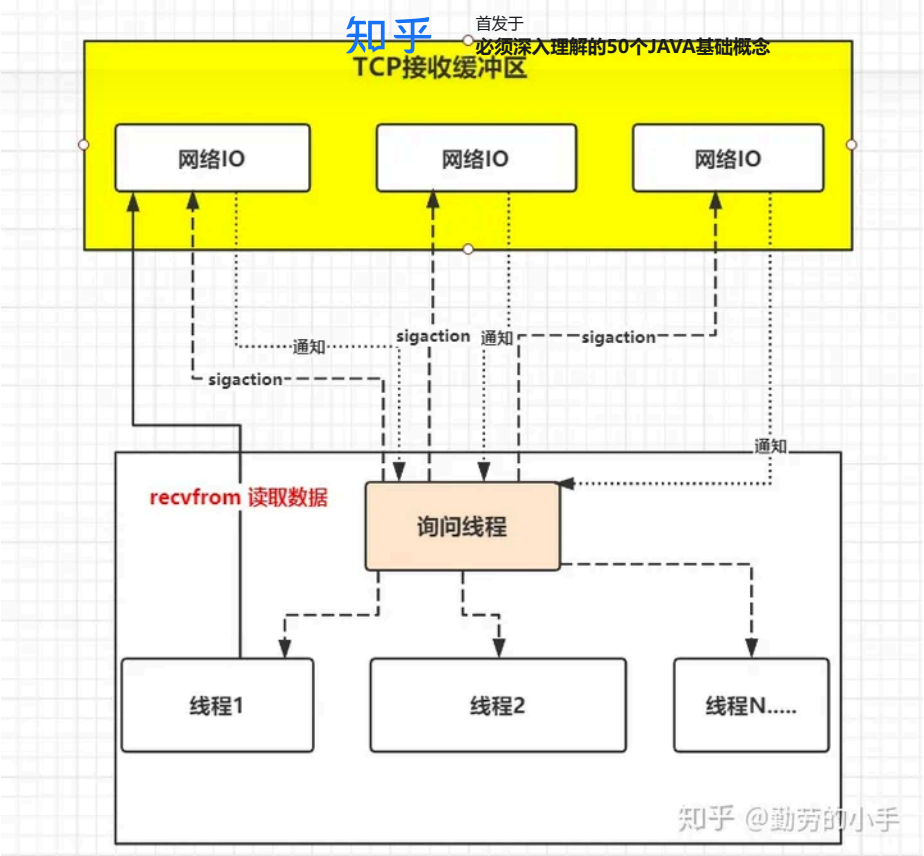
总结：复用IO的基本思路就是通过slect或poll、epoll 来监控多fd ，来达到不必为每个fd创建一个对应的监控线程，从而减少线程资源创建的目的。

信号驱动IO模型

复用IO模型解决了一个线程可以监控多个fd的问题，但是select是采用轮询的方式来监控多个fd的，通过不断的轮询fd的可读状态来知道是否有可读的数据，而无脑的轮询就显得有点暴力，因为大部分情况下的轮询都是无效的，所以有人就想，能不能不要我总是去问你是否数据准备就绪，能不能我发出请求后等你数据准备好了就通知我，所以就衍生了信号驱动IO模型。

于是信号驱动IO不是用循环请求询问的方式去监控数据就绪状态，而是在调用sigaction时候建立一个SIGIO的信号联系，当内核数据准备好之后再通过SIGIO信号通知线程数据准备好后的可读状态，当线程收到可读状态的信号后，此时再向内核发起recvfrom读取数据的请求，因为信号驱动IO的模型下应用线程在发出信号监控后即可返回，不会阻塞，所以这样的方式下，一个应用线程也可以同时监控多个fd。

类似于下图描述：



术语描述：首先开启套接口信号驱动IO功能，并通过系统调用sigaction执行一个信号处理函数，此时请求即刻返回，当数据准备就绪时，就生成对应进程的SIGIO信号，通过信号回调通知应用线程调用recvfrom来读取数据。

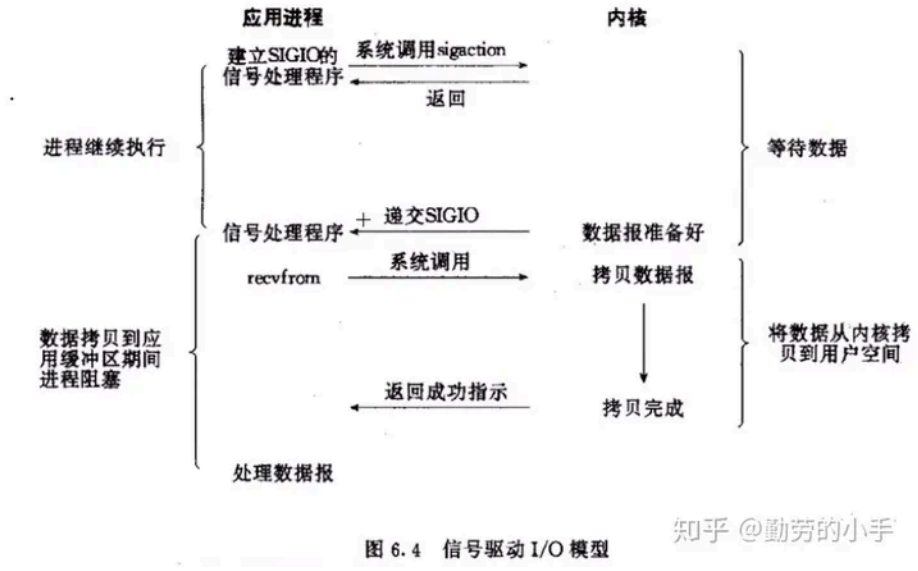


图 6.4 信号驱动 I/O 模型

总结： IO复用模型里面的select虽然可以监控多个fd了，但select其实现的本质上还是通过不断的轮询fd来监控数据状态， 因为大部分轮询请求其实都是无效的，所以信号驱动IO意在通过这种建立信号关联的方式，实现了发出请求后只需要等待数据就绪的通知即可，这样就可以避免大量无效的数据状态轮询操作。

异步IO

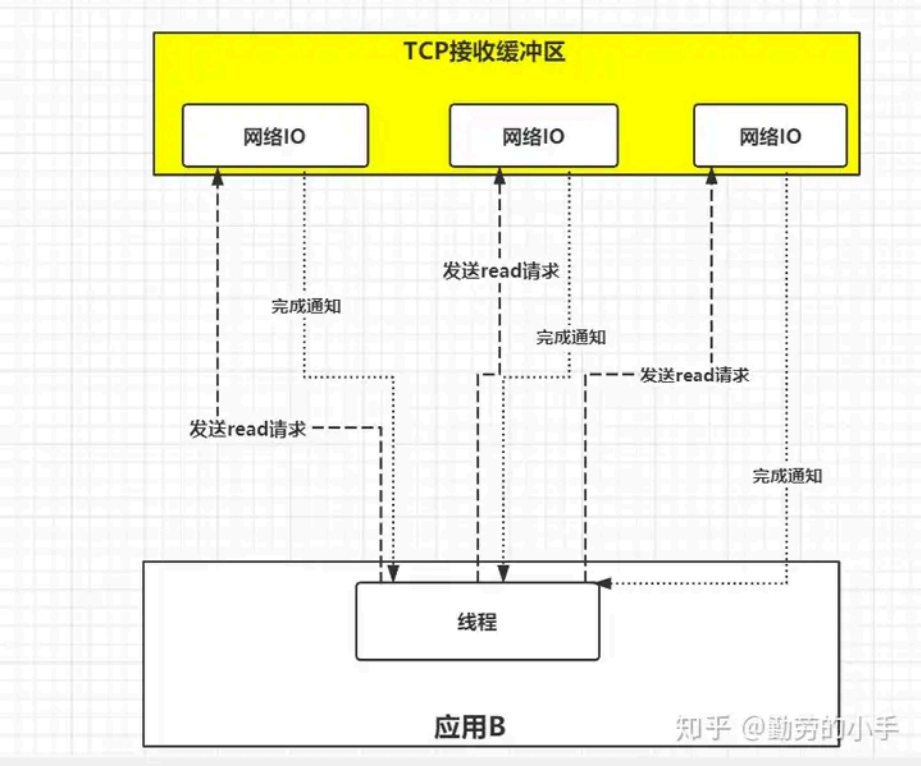
其实经过了上面两个模型的优化，我们的效率有了很大的提升，但是我们当然不会就这样满足了，有没有更好的办法，通过观察我们发现，不管是IO复用还是信号驱动，我们要读取一个数据总是要

发起两阶段的请求，第一次发送select请求，询问数据状态是否准备好，第二次发送recevform请求读取数据。

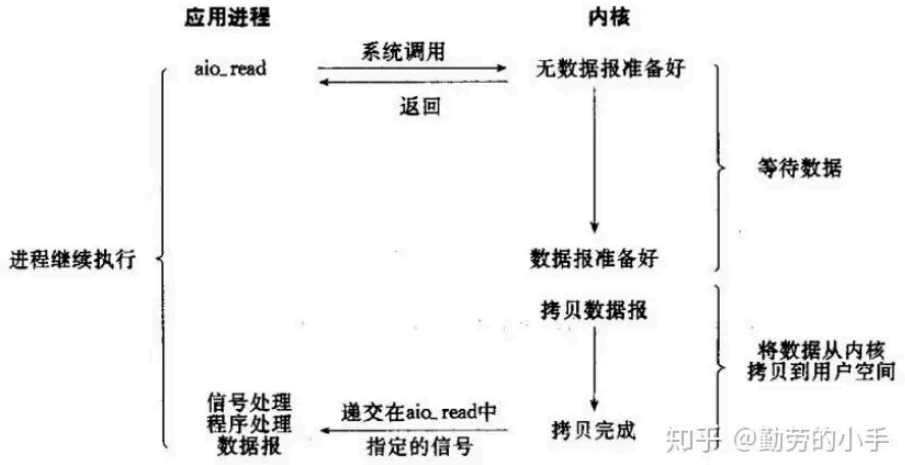
思考一个问题：

也许你一开始就有一个疑问，为什么我们明明是想读取数据，而却非得要先发起一个select询问数据状态的请求，然后再发起真正的读取数据请求,能不能有一种一劳永逸的方式，我只要发送一个请求我告诉内核我要读取数据，然后我就什么都不管了，然后内核去帮我去完成剩下的所有事情?

当然既然你想得出来，那么就会有人做得到，有人设计了一种方案，应用只需要向内核发送一个 read 请求,告诉内核它要读取数据后即刻返回；内核收到请求后会建立一个信号联系，当数据准备就绪，内核会主动把数据从内核复制到用户空间，等所有操作都完成之后，内核会发起一个通知告诉应用，我们称这种一劳永逸的模式为异步IO模型。



术语描述： 应用告知内核启动某个操作，并让内核在整个操作完成之后，通知应用，这种模型与信号驱动模型的主要区别在于，信号驱动IO只是由内核通知我们合适可以开始下一个IO操作，而异步IO模型是由内核通知我们操作什么时候完成。



总结：异步IO的优化思路是解决了应用程序需要先后发送询问请求、发送接收数据请求两个阶段的模式，在异步IO的模式下，只需要向内核发送一次请求就可以完成状态询问和数拷贝的所有操作。

再谈IO模型里面的同步异步

我们通常会说到同步阻塞IO、同步非阻塞IO，异步IO几种术语，通过上面的内容，那么我想你现在肯定已经理解了什么是阻塞什么是非阻塞了，所谓阻塞就是发起读取数据请求的时，当数据还没准备就绪的时候，这时请求是即刻返回，还是在这里等待数据的就绪，如果需要等待的话就是阻塞，反之如果即刻返回就是非阻塞。

我们区分了阻塞和非阻塞后再来分别下同步和异步，在IO模型里面如果请求方从发起请求到数据最后完成的这一段过程中都需要自己参与，那么这种我们就称为同步请求；反之，如果应用发送完指令后就不再参与过程了，只需要等待最终完成结果的通知，那么这就属于异步。

我们再看同步阻塞、同步非阻塞，他们不同的只是发起读取请求的时候一个请求阻塞，一个请求不阻塞，但是相同的是，他们都需要应用自己监控整个数据完成的过程。而为什么只有异步非阻塞而没有异步阻塞呢，因为异步模型下请求指定发送完后就即刻返回了，没有任何后续流程了，所以它注定不会阻塞，所以也就只会有异步非阻塞模型了。

编辑于 2022-12-13 09:21 · IP 属地湖南

网络编程 IO多路复用 NIO

其它的文章看了好久一直模模糊糊，看了楼主的之后清晰多了😁
2022-02-09 回复 2

易悦賀
请教异步IO读取操作交给内核。这样内核的运行负载会高很多吧？
2022-03-25 回复 2

日落耶稣
这篇文章写的很好，回答了一个概念性问题：
select复用出现的逻辑关系：大部分文章都是先介绍socket编程的基本Api（bind/listen/accept）这一套，也就是阻塞模型；随后立马引入select告诉你io多路复用的好处，但问题在于，这部分文章配套的demo引例往往都是在阻塞的情况下对比传统IO和引入select之后的IO，然而在阻塞情况下这两种情况并没有什么不同（传统io使用accept接受每个连接，处理完一个连接之后才能处理下一个，select则将准备好的连接以数组方式返回，用户层仍需要遍历每一个fd，并且处理每一个fd也是串行的，导致这部分例子并没有说清楚select究竟好在哪，使读者产生疑问）
以io模型的角度来引入多路复用我认为是最好的方式，因为select只有在非阻塞多线程时才能看出多路复用的优势何在。
2023-07-31 回复 1

涵涵涵
讲的真的很不错👍
2020-04-17 回复 1

贾亮
感觉是看过的最容易懂的一篇了
2021-06-15 回复 5

宇轩笔墨
除了扣666666，我还能做什么😭😭😭
2021-09-09 回复 2

点击查看全部评论 ›

理性发言，友善互动

文章被以下专栏收录

必须深入理解的50个JAVA基础概念
你和高手之间差的就是这些。

java

优质文章收录
本专栏记录自己在学习过程中阅读到的优质文章。

推荐阅读

粗糙集模型
我爱我家岑姑娘

大模型偏好对齐-DPO
Linsi... 发表于NLP-L...

从零手搓大模型
三十三天