简明网络I/O模型---同步异步阻塞非阻塞之惑


作者 人世间 (/u/5qrPPM) + 关注
2015.04.19 23:49* 字数 2582 阅读 5135 评论 3 喜欢 22
(/u/5qrPPM)

网络I/O模型

人多了，就会有问题。web刚出现的时候，光顾的人很少。近年来网络应用规模逐渐扩大，应用的架构也需要随之改变。C10k的问题，让工程师们需要思考服务的性能与应用的并发能力。

网络应用需要处理的无非就是两大类问题，网络**I/O**，数据计算。相对于后者，网络I/O的延迟，给应用带来的性能瓶颈大于后者。网络I/O的模型大致有如下几种：

- 同步模型（synchronous I/O）
    - 阻塞I/O（bloking I/O）
    - 非阻塞I/O（non-blocking I/O）
    - 多路复用I/O（multiplexing I/O）
    - 信号驱动式I/O（signal-driven I/O）
- 异步I/O（asynchronous I/O）

网络I/O的本质是socket的读取，socket在linux系统被抽象为流，I/O可以理解为对流的操作。这个操作又分为两个阶段：

1. 等待流数据准备（wating for the data to be ready）。
2. 从内核向进程复制数据（copying the data from the kernel to the process）。

对于socket流而已，

- 第一步通常涉及等待网络上的数据分组到达，然后被复制到内核的某个缓冲区。
- 第二步把数据从内核缓冲区复制到应用进程缓冲区。

I/O模型

举个简单比喻，来了解这几种模型。网络IO好比钓鱼，等待鱼上钩就是网络中等待数据准备好的过程，鱼上钩了，把鱼拉上岸就是内核复制数据阶段。钓鱼的人就是一个应用进程。

阻塞I/O（bloking I/O）

阻塞I/O是最流行的I/O模型。它符合人们最常见的思考逻辑。阻塞就是进程 **"被"** 休息，**CPU**处理其它进程去了。在网络I/O的时候，进程发起 `recvform` 系统调用，然后进程就被阻塞了，什么也不干，直到数据准备好，并且将数据从内核复制到用户进程，最后进程再处理数据，在等待数据到处理数据的两个阶段，整个进程都被阻塞。不能处理别的网络I/O。大致如下图：

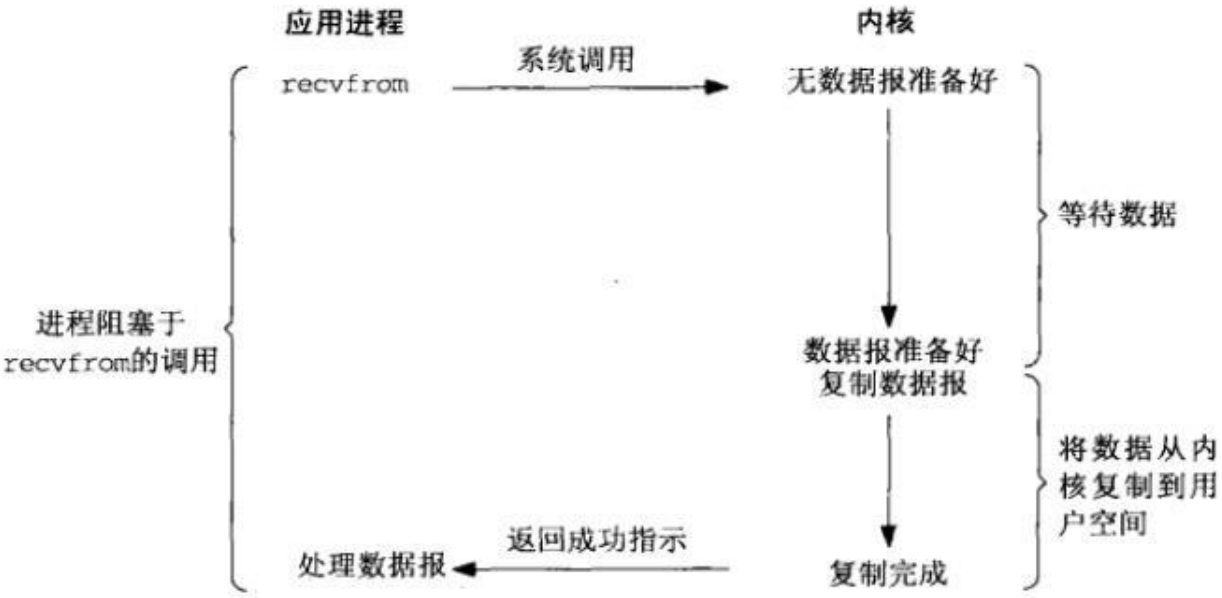


图6-1　阻塞式I/O模型

1.png

这就好比我们去钓鱼，抛竿之后就一直在岸边等，直到等待鱼上钩。然后再一次抛竿，等待下一条鱼上钩，等待的时候，什么事情也不做，大概会胡思乱想吧。

> 阻塞IO的特点就是在IO执行的两个阶段都被block了

非阻塞I/O（non-bloking I/O）

在网络I/O时候，非阻塞I/O也会进行recvform系统调用，检查数据是否准备好，与阻塞I/O不一样，"非阻塞将大的整片时间的阻塞分成N多的小的阻塞, 所以进程不断地有机会'被' CPU光顾"。

也就是说非阻塞的recvform系统调用调用之后，进程并没有被阻塞，内核马上返回给进程，如果数据还没准备好，此时会返回一个error。进程在返回之后，可以干点别的事情，然后再发起recvform系统调用。重复上面的过程，循环往复的进行recvform系统调用。这个过程通常被称之为 轮询 。轮询检查内核数据，直到数据准备好，再拷贝数据到进程，进行数据处理。需要注意，拷贝数据整个过程，进程仍然是属于阻塞的状态。

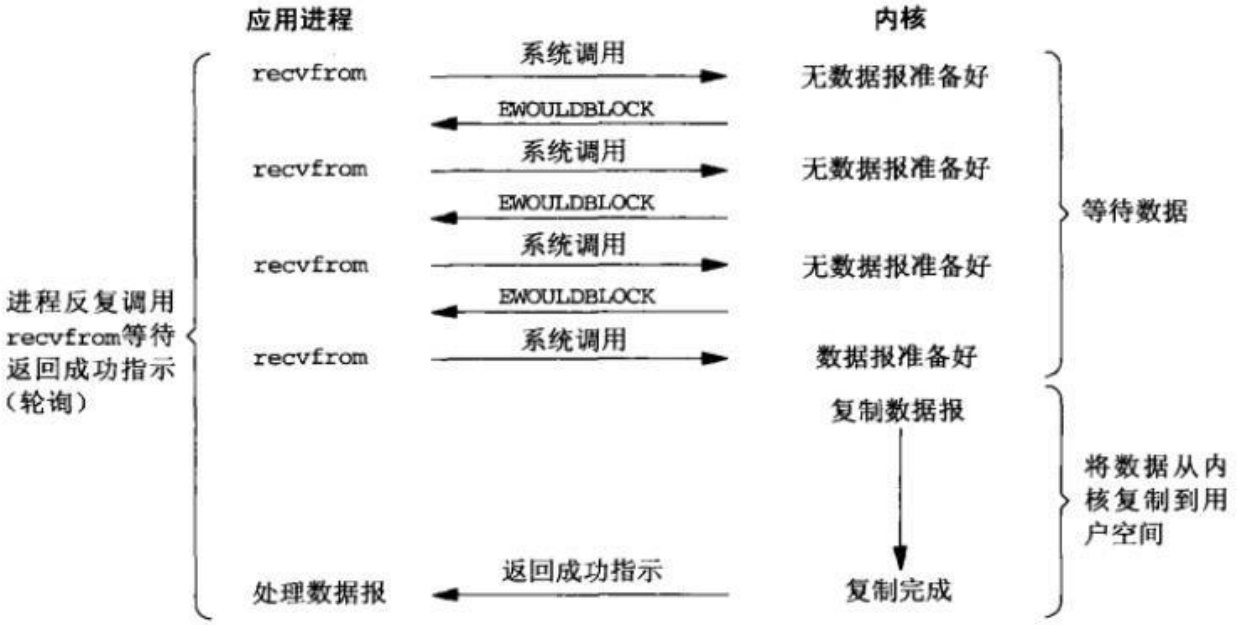


图6-2　非阻塞式I/O模型

2.png

我们再用钓鱼的方式来类别，当我们抛竿入水之后，就看下鱼漂是否有动静，如果没有鱼上钩，就去干点别的事情，比如再挖几条蚯蚓。然后不久又来看看鱼漂是否有鱼上钩。这样往返的检查又离开，直到鱼上钩，再进行处理。

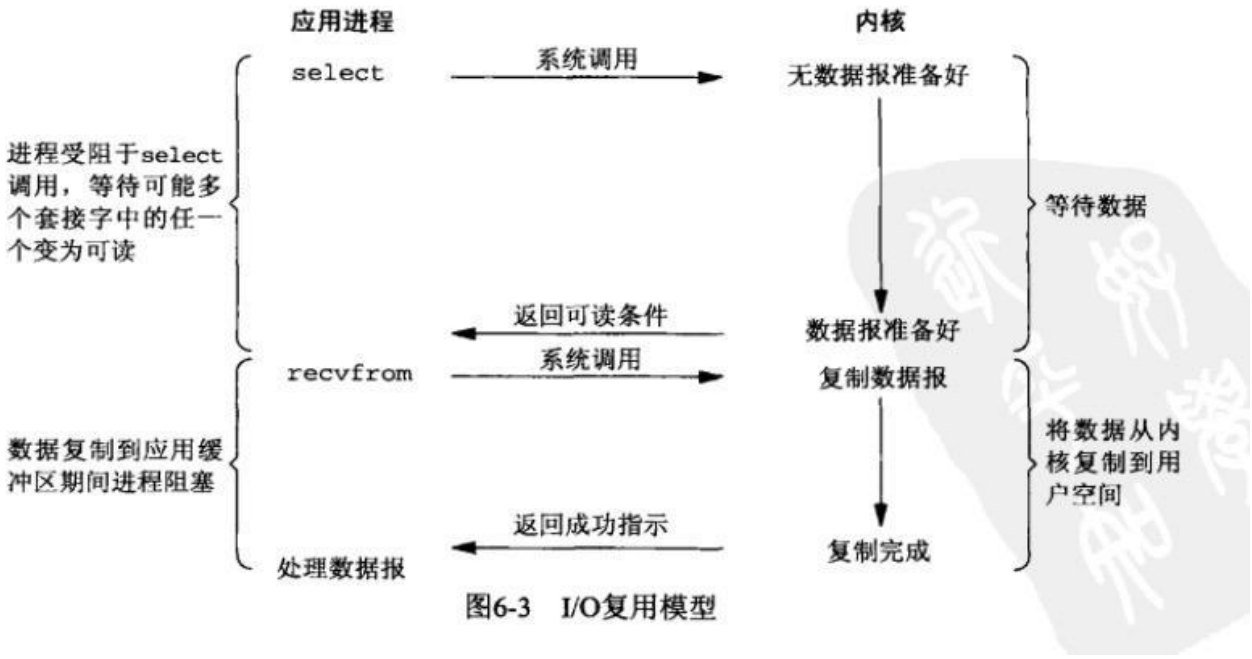
> 非阻塞 IO的特点是用户进程需要不断的主动询问kernel数据是否准备好。

多路复用I/O（multiplexing I/O）

可以看出，由于非阻塞的调用，轮询占据了很大一部分过程，轮询会消耗大量的CPU时间。结合前面两种模式。如果轮询不是进程的用户态，而是有人帮忙就好了。多路复用正好处理这样的问题。

多路复用有两个特别的系统调用 `select` 或 `poll` 。select调用是内核级别的，select轮询相对非阻塞的轮询的区别在于---前者可以等待多个socket，当其中任何一个socket的数据准好了，就能返回进行可读，然后进程再进行recvform系统调用，将数据由内核拷贝到用户进程，当然这个过程是阻塞的。多路复用有两种阻塞，select或poll调用之后，会阻

塞进程，与第一种阻塞不同在于，此时的select不是等到socket数据全部到达再处理, 而是有了一部分数据就会调用用户进程来处理。如何知道有一部分数据到达了呢？监视的事情交给了内核，内核负责数据到达的处理。也可以理解为"非阻塞"吧。



3.png

对于多路复用，也就是轮询多个socket。钓鱼的时候，我们雇了一个帮手，他可以同时抛下多个钓鱼竿，任何一杆的鱼一上钩，他就会拉杆。他只负责帮我们钓鱼，并不会帮我们处理，所以我们还得在一帮等着，等他把收杆。我们再处理鱼。多路复用既然可以处理多个I/O，也就带来了新的问题，多个I/O之间的顺序变得不确定了，当然也可以针对不同的编号。

> 多路复用的特点是通过一种机制一个进程能同时等待IO文件描述符，内核监视这些文件描述符（套接字描述符），其中的任意一个进入读就绪状态，select，poll，epoll函数就可以返回。对于监视的方式，又可以分为 select， poll， epoll三种方式。

了解了前面三种模式，在用户进程进行系统调用的时候，他们在等待数据到来的时候，处理的方式不一样，直接等待，轮询，select或poll轮询，第一个过程有的阻塞，有的不阻塞，有的可以阻塞又可以不阻塞。当时第二个过程都是阻塞的。从整个I/O过程来看，他们都是顺序执行的，因此可以归为同步模型(asynchronous)。都是进程主动向内核检查。

异步I/O（asynchronous I/O）

相对于同步I/O，异步I/O不是顺序执行。用户进程进行 aio_read 系统调用之后，无论内核数据是否准备好，都会直接返回给用户进程，然后用户态进程可以去做别的事情。等到socket数据准备好了，内核直接复制数据给进程，然后从内核向进程发送通知。I/O两个阶段，进程都是非阻塞的。

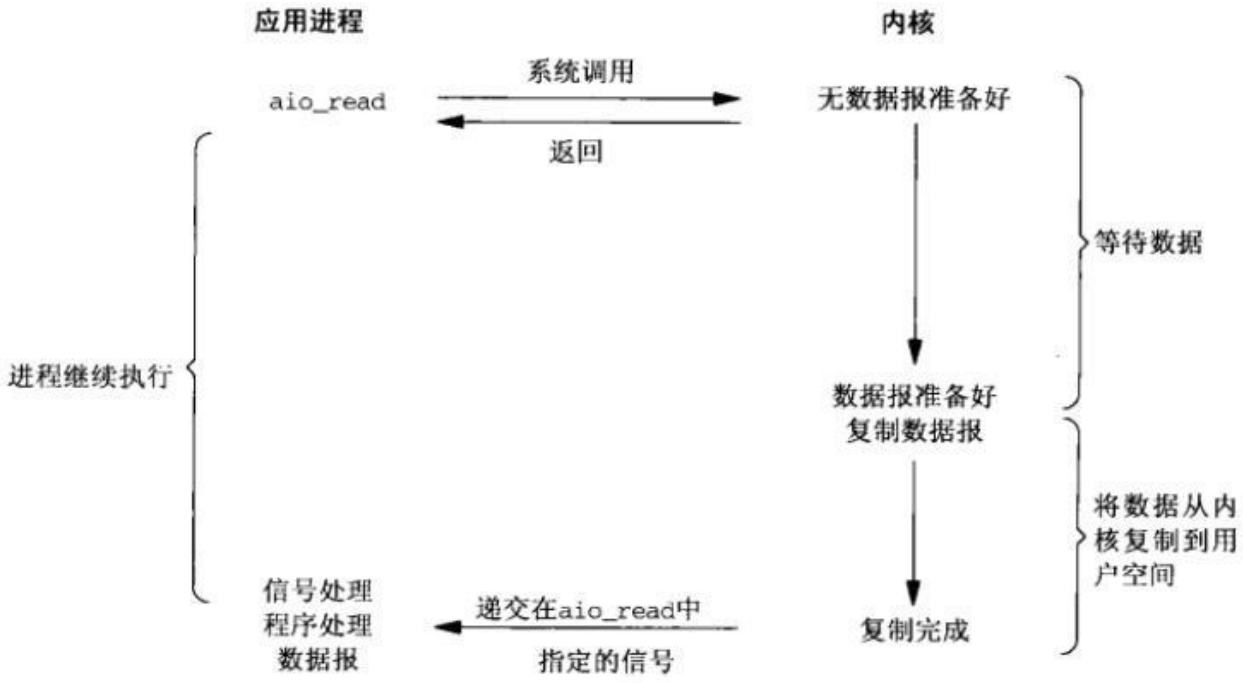


图6-5　异步I/O模型

4.png

比之前的钓鱼方式不一样，这一次我们雇了一个钓鱼高手。他不仅会钓鱼，还会在鱼上钩之后给我们发短信，通知我们鱼已经准备好了。我们只要委托他去抛竿，然后就能跑去干别的事情了，直到他的短信。我们再回来处理已经上岸的鱼。

同步和异步的区别

通过对上述几种模型的讨论，需要区分阻塞和非阻塞，同步和异步。他们其实是两组概念。区别前一组比较容易，后一种往往容易和前面混合。对于同步和异步而言，往往是一个函数调用之后，是否直接返回结果，如果函数挂起，直到获得结果，这是同步；如果函数马上返回，等数据到达再通知函数，那么这是异步的路程。

至于阻塞和非阻塞，则是函数是否让线程挂起不再往下执行。通常同步阻塞，异步非阻塞。什么情况下是异步阻塞呢？即函数调用之后并没有返回结果而注册了回调函数，非阻塞的情况下，函数也马上返回，可是如果此时函数不返回，那么此时就是阻塞的状态，等数据到达通知函数，依然是异步的过程。

区分阻塞和非阻塞只要区分函数调用之后是否挂起返回就可以了，区分异步和同步，则是函数调用之后，数据或条件满足之后如何通知函数。等待数据返回则是同步，通过回调则是异步。

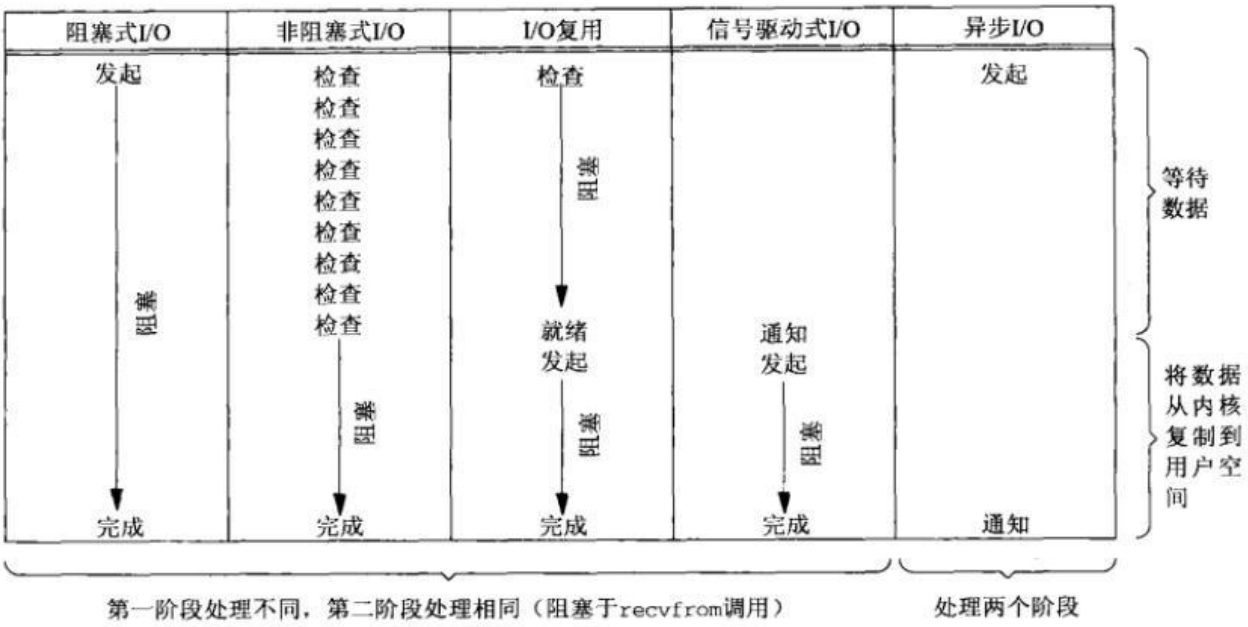


图6-6　5种I/O模型的比较

5.png

对于同步模型，主要是第一阶段处理方法不一样。而异步模型，两个阶段都不一样。这里我们忽略了信号驱动模式。这几个名词还是容易让人迷惑。

本文所讨论的IO模型来自大名鼎鼎的《unix网络编程：卷1套接字联网API》。单台服务器中的linux系统。分布式的环境或许会不一样。个人学习笔记，参考了网络上大多数文章，做了一点小测试。