【系统编程】五种IO模型分析

码农有道 2018-05-23 07:58

> 作者：骏马金龙
> 链接：http://www.cnblogs.com/f-ck-need-u/
> 码农有道作了部分修改

码农有道

历史文章目录（请戳我）

关于码农有道（请戳我）

在网络数据传输时经历了哪些buffer（请戳我）一文中主要总结了从客户端发起一个http请求，网络数据的流向，有了上文的基础，本文再来讲讲五种I/O模型。

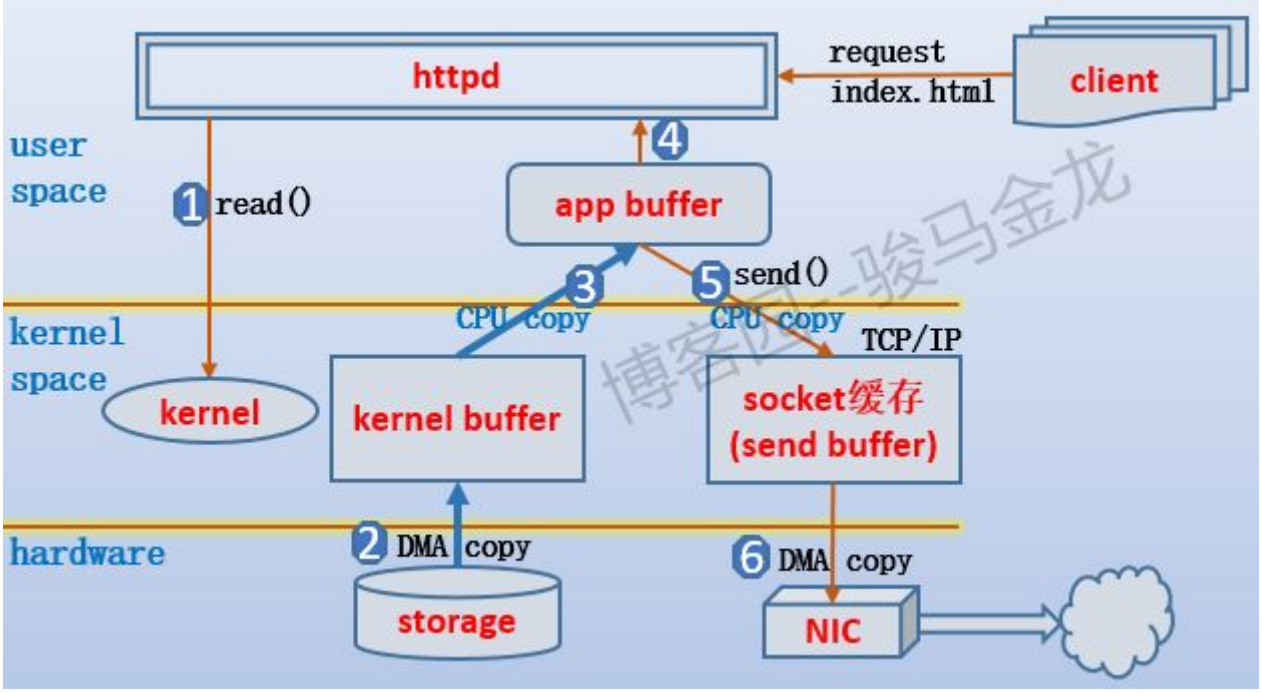
所谓的IO模型，描述的是出现I/O等待时进程的状态以及处理数据的方式。围绕着进程的状态、数据准备到kernel buffer再到app buffer的两个阶段展开。其中数据复制到kernel buffer的过程称为数据准备阶段，数据从kernel buffer复制到app buffer的过程称为数据复制阶段。请记住这两个概念，后面描述I/O模型时会一直用这两个概念。

本文以httpd进程的TCP连接方式处理本地文件为例，请无视httpd是否真的实现了如此、那般的功能，也请无视TCP连接处理数据的细节，这里仅仅只是作为方便解释的示例而已。另外，本文用本地文件作为I/O模型的对象不是很适合，它的重头戏是在套接字上，

再次说明：从硬件设备到内存的数据传输过程是不需要CPU参与的，而内存间传输数据是需要CPU参与的。

Blocking I/O模型

如图：



假设客户端发起index.html的文件请求，httpd需要将index.html的数据从磁盘中加载到自己的httpd app buffer中，然后复制到send buffer中发送出去。

但是在httpd想要加载index.html时，它首先检查自己的app buffer中是否有index.html对应的数据，没有就发起系统调用让内核去加载数据，例如read()，内核会先检查自己的kernel buffer中是否有index.html对应的数据，如果没有，则从磁盘中加载，然后将数据准备到kernel buffer，再复制到app buffer中，最后被httpd进程处理。

如果使用Blocking I/O模型：

1：当设置为blocking i/o模型，httpd从1到3都是被阻塞的。

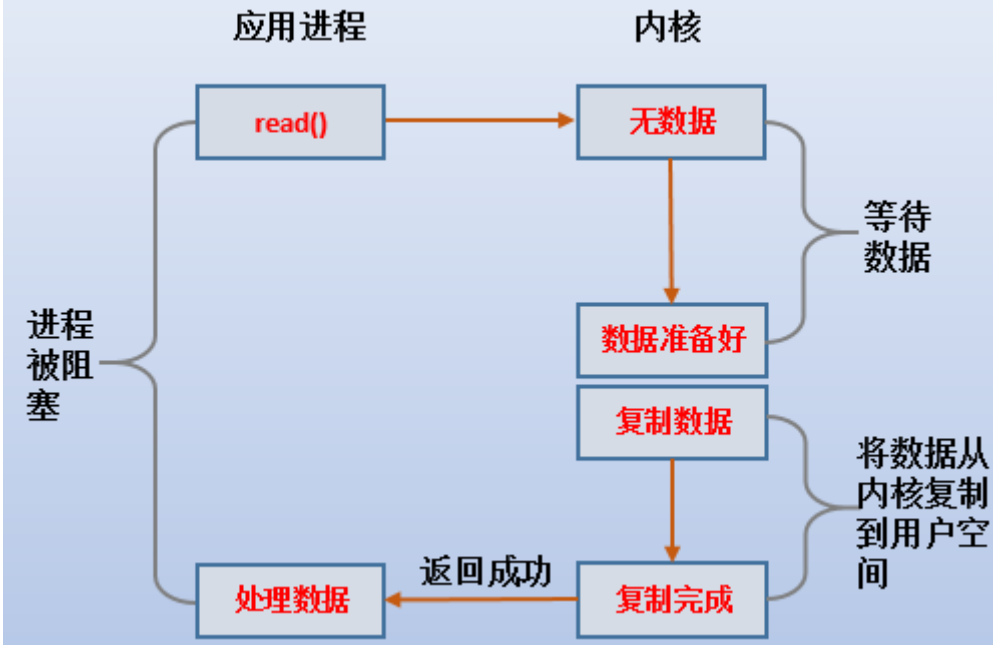
2：只有当数据复制到app buffer完成后，或者发生了错误，httpd才被唤醒处理它app buffer中的数据。

3：cpu会经过两次上下文切换：用户空间到内核空间再到用户空间。

4：由于2阶段的拷贝是不需要CPU参与的，所以在2阶段准备数据的过程中，cpu可以去处理其它进程的任务。

5：3阶段的数据复制需要CPU参与，将httpd阻塞，在某种程度上来说，有助于提升它的拷贝速度。

如下图：



Non-Blocking I/O模型

如果使用Non-Blocking I/O模型：

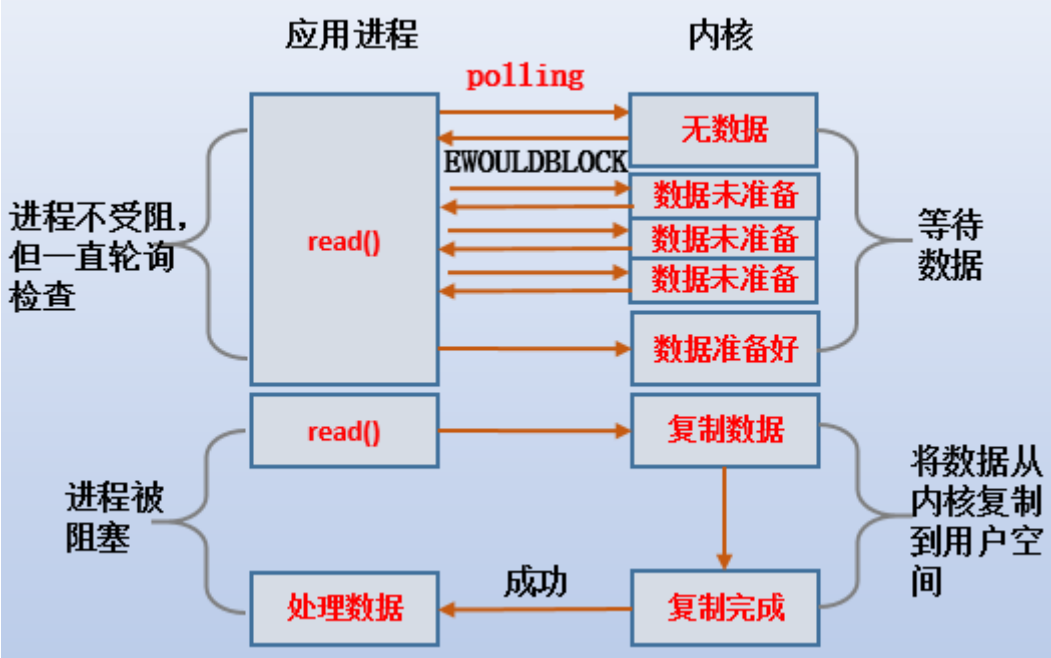
1：.当设置为non-blocking时，httpd第一次发起系统调用(如read())后，立即返回一个错误值EWOULDBLOCK(至于read()读取一个普通文件时能否返回EWOULDBLOCK请无视，毕竟I/O模型主要是针对套接字文件的，就当read()是recv()好了)，而不是让httpd进入睡眠状态。

2：虽然read()立即返回了，但httpd还要不断地去发送read()检查内核：数据是否已经成功拷贝到kernel buffer了？这称为轮询(polling)。每次轮询时，只要内核没有把数据准备好，read()就返回错误信息EWOULDBLOCK。

3：直到kernel buffer中数据准备完成，再去轮询时不再返回EWOULDBLOCK，而是将httpd阻塞，以等待数据复制到app buffer。

4：httpd在1到2阶段不被阻塞，但是会不断去发送read()轮询。在3被阻塞，将cpu交给内核把数据copy到app buffer。

如下图：



I/O Multiplexing

称为多路IO模型或IO复用，意思是可以检查多个IO等待的状态。有三种IO复用模型：**select**、**poll**和**epoll**。其实它们都是一种函数，用于监控指定文件描述符的数据是否就绪，就绪指的是对某个系统调用不再阻塞了，例如对于read()来说，就是数据准备好了就是就绪状态。

就绪种类包括是否可读、是否可写以及是否异常，其中可读条件中就包括了数据是否准备好。当就绪之后，将通知进程，进程再发送对数据操作的系统调用，如read()。所以，这三个函数仅仅只是处理了数据是否准备好以及如何通知进程的问题。可以将这几个函数结合阻塞和非阻塞IO模式使用，例如设置为非阻塞时，select()/poll()/epoll将不会阻塞在对应的描述符上，调用函数的进程/线程也就不会被阻塞。

如果使用I/O Multiplexing模型：

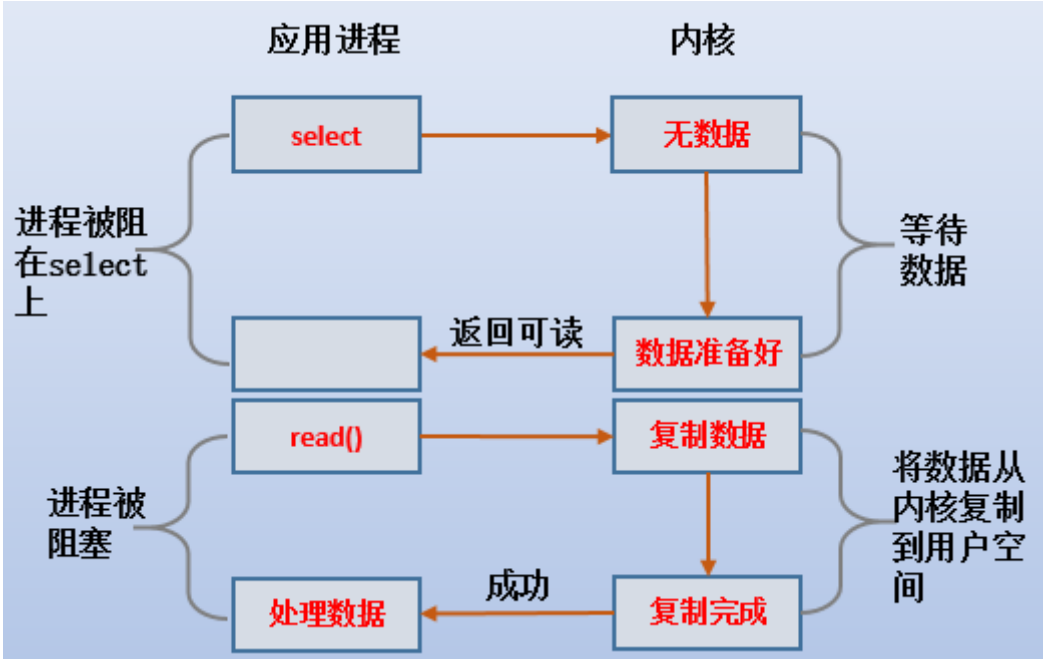
1：当想要加载某个文件时，假如httpd要发起read()系统调用，如果是阻塞或者非阻塞情形，那么read()会根据数据是否准备好而决定是否返回，是否可以主动去监控这个数据是否准备到了kernel buffer中呢，亦或者是否可以监控send buffer中是否有新数据进入呢？这就是select()/poll()/epoll的作用。

2：当使用select()时，httpd发起一个select调用，然后httpd进程被select()"阻塞"。由于此处假设只监控了一个请求文件，所以select()会在数据准备到kernel buffer中时直接唤醒httpd进程。之所以阻塞要加上双引号，是因为select()有时间间隔选项可用控制阻塞时长，如果该选项设置为0，则select不阻塞，此时表示立即返回但一直轮询检查是否就绪，还可以设置为永久阻塞。

3：当select()的监控对象就绪时，将通知(轮询情况)或唤醒(阻塞情况)httpd进程，httpd再发起read()系统调用，此时数据会从kernel buffer复制到app buffer中并read()成功。

4：httpd发起第二个系统调用(即read())后被阻塞，CPU全部交给内核用来复制数据到app buffer。(5).对于httpd只处理一个连接的情况下，IO复用模型还不如blocking I/O模型，因为它前后发起了两个系统调用(即select()和read())，甚至在轮询的情况下会不断消耗CPU。但是IO复用的优势就在于能同时监控多个文件描述符。

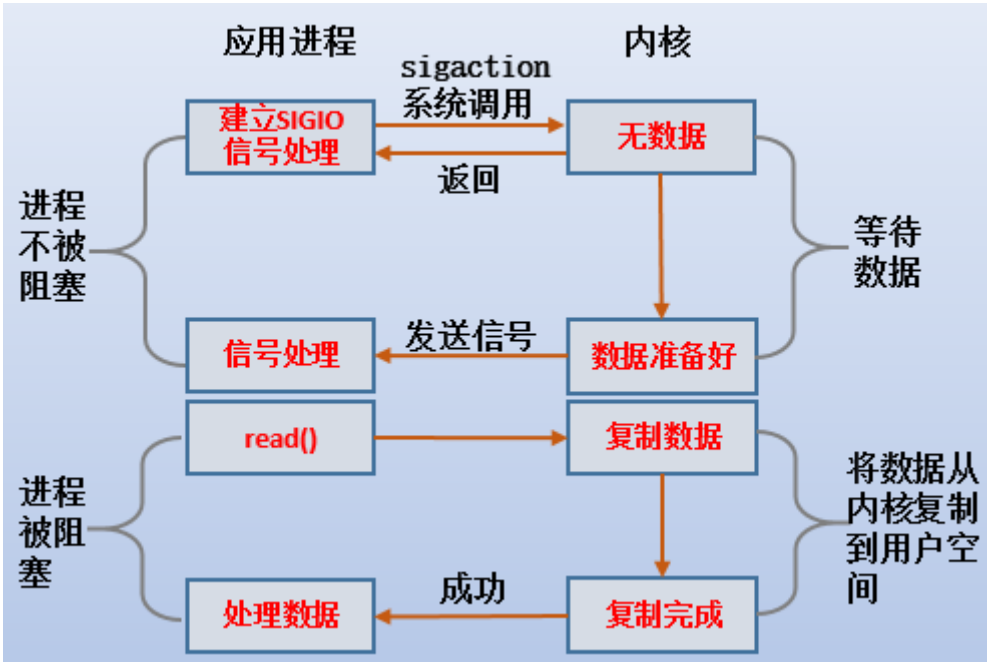
如图：



Signal-driven I/O模型

即信号驱动IO模型。当开启了信号驱动功能时，首先发起一个信号处理的系统调用，如sigaction()，这个系统调用会立即返回。但数据在准备好时，会发送SIGIO信号，进程收到这个信号就知道数据准备好了，于是发起操作数据的系统调用，如read()。

在发起信号处理的系统调用后，进程不会被阻塞，但是在read()将数据从kernel buffer复制到app buffer时，进程是被阻塞的。如图：

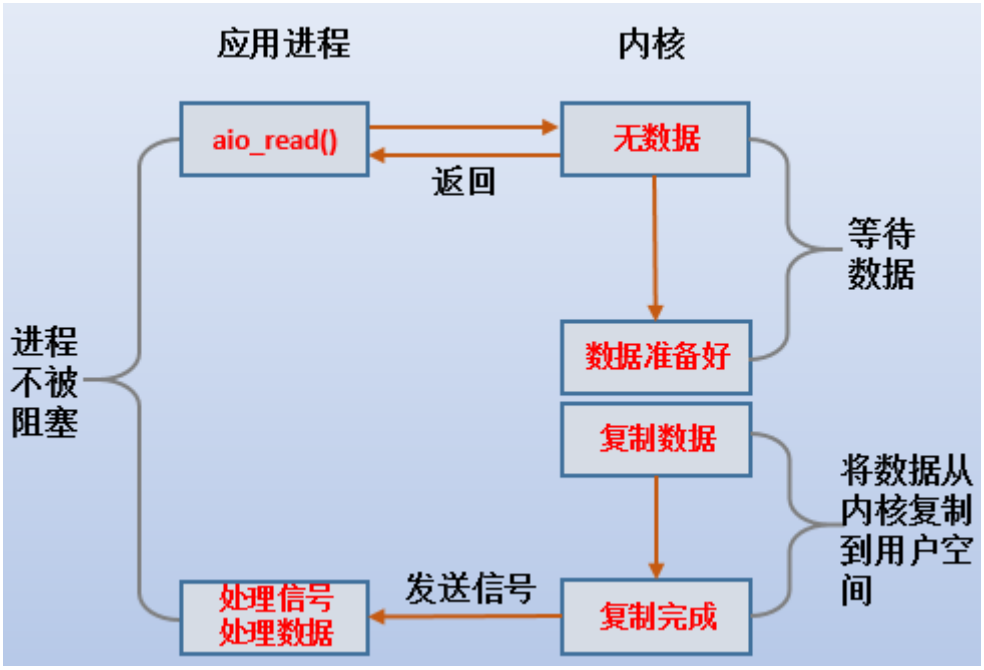


Asynchronous I/O模型

即异步IO模型。当设置为异步IO模型时，httpd首先发起异步系统调用(如aio_read()，aio_write()等)，并立即返回。这个异步系统调用告诉内核，不仅要准备好数据，还要把数据复制到app buffer中。

httpd从返回开始，直到数据复制到app buffer结束都不会被阻塞。当数据复制到app buffer结束，将发送一个信号通知httpd进程。

如图：



看上去异步很好，但是注意，在复制kernel buffer数据到app buffer中时是需要CPU参与的，这意味着不受阻的httpd会和异步调用函数争用CPU。如果并发量比较大，httpd接入的连接数可能就越多，CPU争用

情况就越严重，异步函数返回成功信号的速度就越慢。如果不能很好地处理这个问题，异步IO模型也不一定就好。

同步I/O与异步I/O，阻塞与非阻塞区别

阻塞、非阻塞、IO复用、信号驱动都是同步IO模型。因为在发起操作数据的系统调用(如本文的read())过程中是被阻塞的。这里要注意，虽然在加载数据到kernel buffer的数据准备过程中可能阻塞、可能不阻塞，但kernel buffer才是read()函数的操作对象，同步的意思是让kernel buffer和app buffer数据同步。显然，在保持kernel buffer和app buffer同步的过程中，进程必须被阻塞，否则read()就变成异步的read()。

只有异步IO模型才是异步的，因为发起的异步类的系统调用(如aio_read())已经不管kernel buffer何时准备好数据了，就像后台一样read一样，aio_read()可以一直等待kernel buffer中的数据，在准备好了之后，aio_read()自然就可以将其复制到app buffer。

如图：