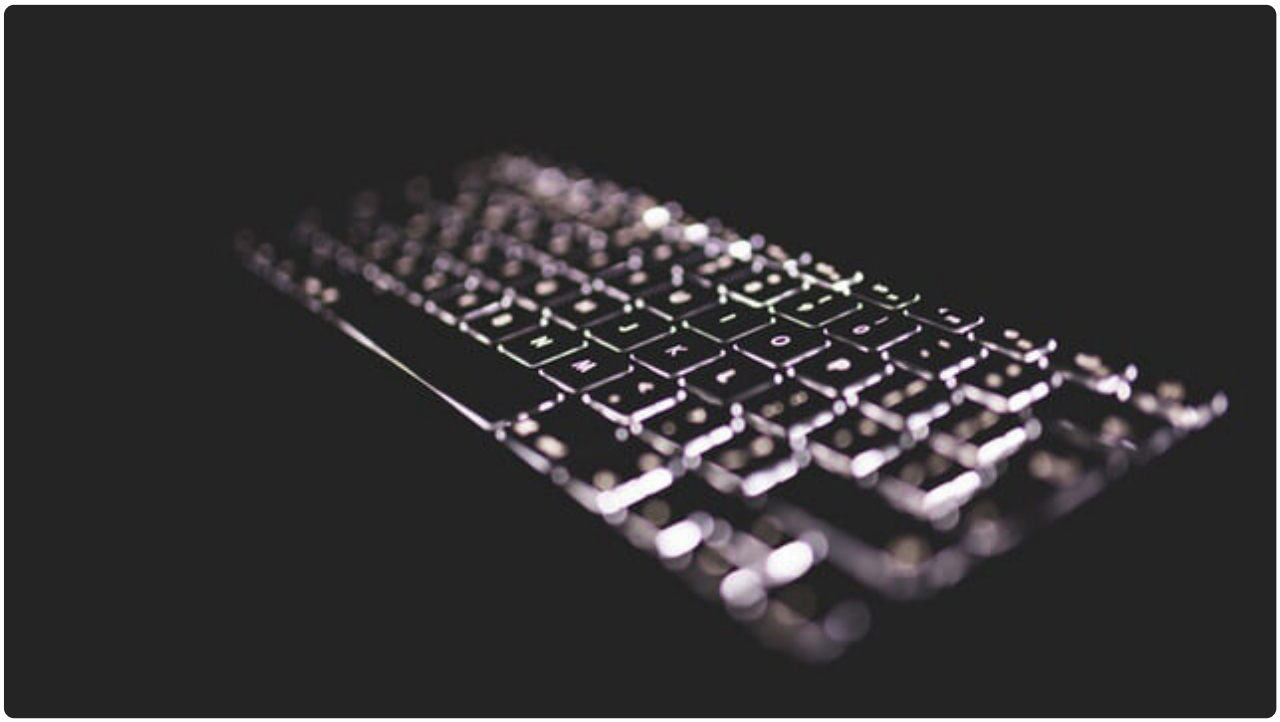
# 05 Linux 的 IO 原理

更新时间：2020-01-13 09:57:23



> 老骥伏枥，志在千里；烈士暮年，壮心不已。——曹操

**前言**

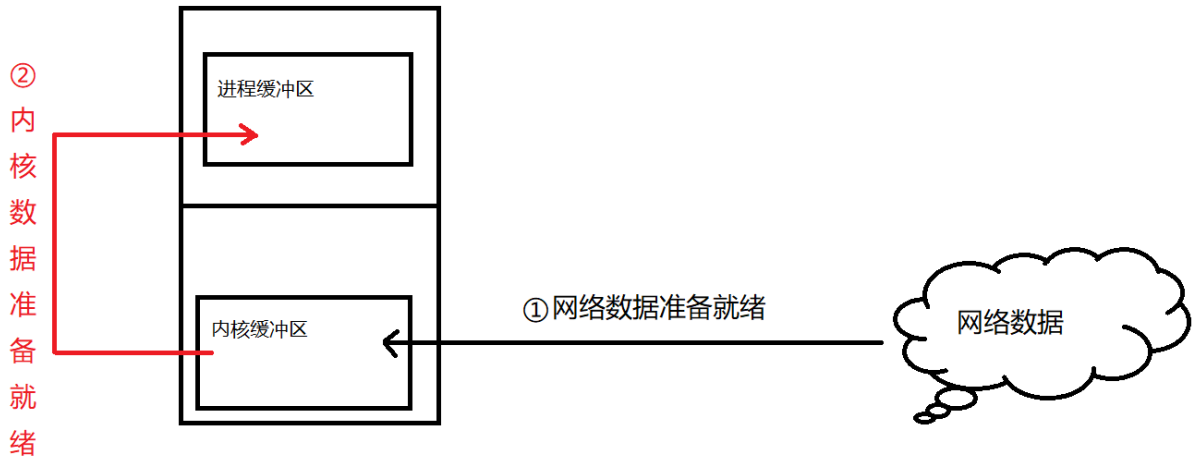
在介绍 Nginx 的其它内容之前，我觉得有必要和大家分享一下 **I/O模型**，理解这一部分内容才能真正理解 Nginx 为什么高效。网络上面有很多 I/O 模型相关的内容，好像每个人都能说上那么几句，但是很多人的内容都经不起推敲，我们只需要稍微追问上几句，他们就很难自圆其说了。所以，我们这里会分享一下我对这部分内容的理解。

> 哈哈，如果觉得哪里有问题的话，可以在评论区给我留言～～～

**相关概念**

在介绍 I/O 模型之前，我们要首先要知道什么是 I/O？

I/O 是 Input/Output 的缩写，指操作系统中的输入输出操作，从网络中获取或者发送数据也属于 I/O 操作的一种。我们以网络数据为例看一下整个 I/O 的流程：




我们从图中可以看到，其实整个过程分为两步：

①：内核等待网络数据；

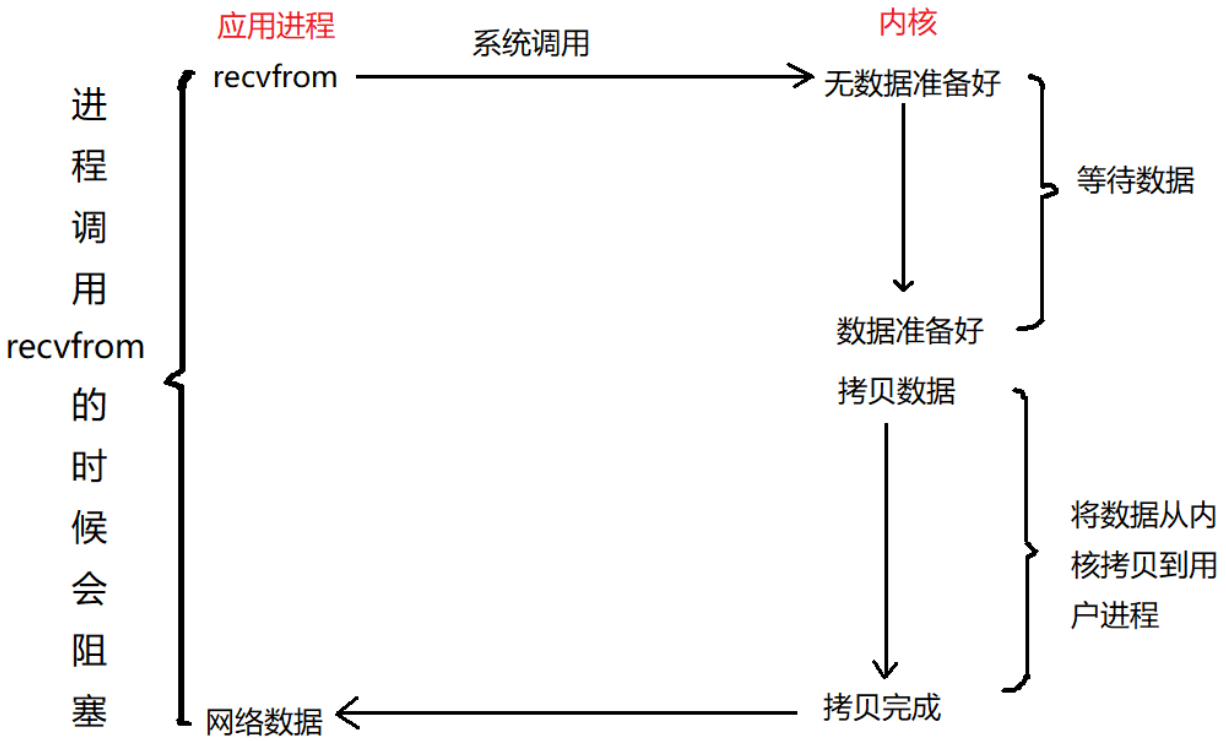
②：用户进程将数据从内核缓冲区中读取出来。

> 大家一定要记住这两个过程，这是后面分析 I/O 模型时候的基础。

**I/O 模型**

在 Linux 中，分了五种 I/O 模型，如下图所示：

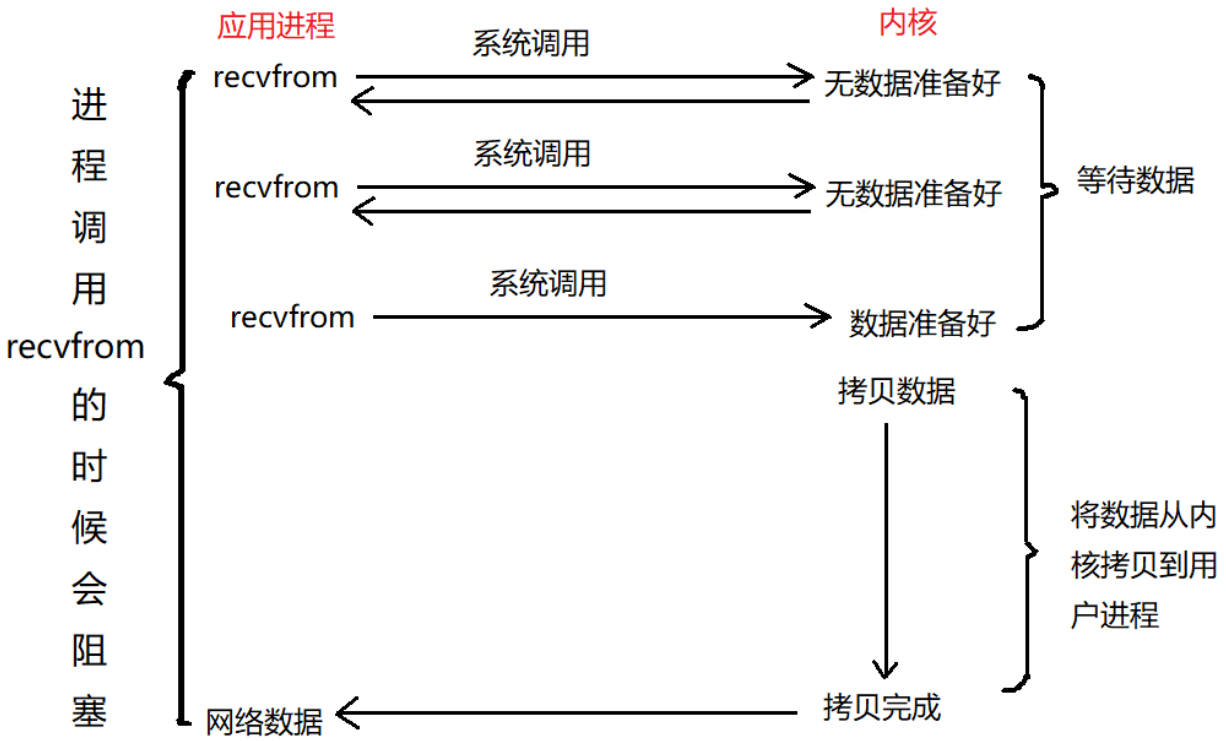

## 阻塞 I/O




用户进程调用 recvfrom 函数获取数据，这个时候如果内核没有数据，那么用户进程会一直**阻塞在等待数据阶段**，直到数据准备好之后，将数据从内核缓冲区中拷贝到用户缓冲区中。在等待数据的时候，因为用户进程是阻塞的，所以不能做其他的事情。

这就好比你去餐厅吃饭，你一直在后厨等着，知道你的饭菜做好之后才去饭桌吃饭。在等待的期间，你什么事情都不能干。

### 非阻塞 I/O




用户进程通过 recvfrom 从内核获取数据的时候，如果内核的数据没有准备好，用户进程就直接返回，去干别的事情，过一会儿重新调用 recvfrom，直到内核数据准备好。

还是以餐厅就餐为例：

**我：** 服务员，你好，请问我点的餐好了吗？

**服务员：** 先生，您的餐还没有准备好，请稍等。

… 五分钟后…

**我：** 服务员，你好，请问我点的餐好了吗？

**服务员：** 先生，您的餐马上就好

… 五分钟后…

**我：** 服务员，你好，请问我点的餐好了吗？

**服务员：** 先生，您的餐已经做好了，现在就给您取过来。

每次我问完服务员之后，如果餐尚未做好，我就可以干其他的事情了，比如看会手机，抽支烟，等过几分钟再问一次，直到餐品做好为止。

## I/O 多路复用

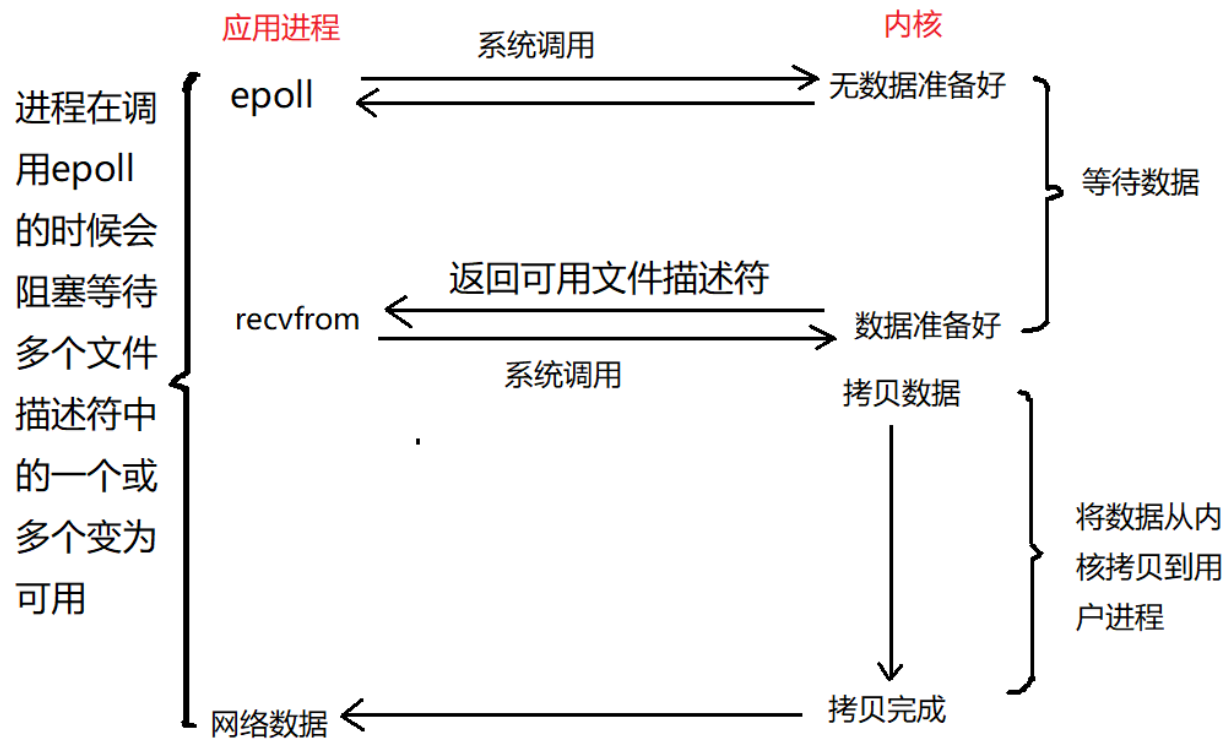
这里还是以餐厅的场景为例，大家想一下我们平时去餐厅，每个餐厅的服务员数量和顾客数量哪个更多呢？

答案是显而易见的，肯定是顾客数量多啊。那么这是为啥呢，为什么不为每一个顾客安排一位服务员呢？因为给每一位顾客分配一位服务员的人工成本太高了。

> 餐厅老板：有请那么多服务员的钱我买两斤排骨它不香吗？

当然，还有一个原因就是：在某一个时间段内，并不是所有的顾客都需要服务员，只是部分顾客才需要，所以我们并不需要为每一个顾客安排一个服务员。

在操作系统中也有同样的机制，同时监控很多个 I/O 操作，当一个 I/O 的数据准备好之后，就进行数据的拷贝.




我们对比一下 I/O 多路复用和餐厅的例子：

epoll 函数就相当于 服务员；

一个 I/O 操作就相当于一个 顾客；

每个 服务员 服务很多个 顾客，对应于一个 epoll 监控多个 I/O 操作；

如果同时有多个 顾客 有需求，就对应于同时有多个 I/O 的数据准备好了，这时候 服务员 要逐个的帮助顾客解决问题，同样的， 操作系统 也是逐个的处理每个 I/O。

讲到这里，大家应该就明白什么是 I/O 多路复用了吧，其实就是同时监控多个 I/O 操作。

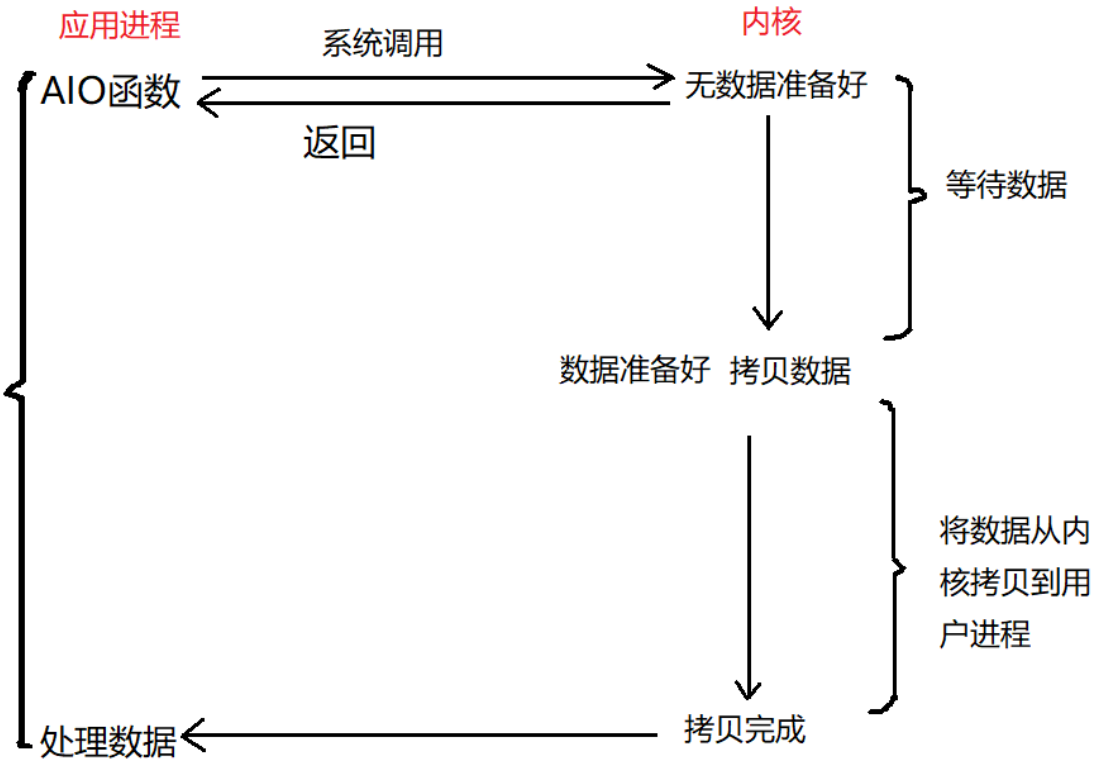
Nginx 就是使用了 I/O 多路复用，所以能够达到非常高的并发量。

> 其实计算机中的很多功能都是从生活中抽象出来的，我们通过类比生活中的经验，可以很容易的理解计算机

## 异步 I/O

我们仍然以餐厅的场景为例来说明异步 I/O 的过程。

服务员告诉我还要再等一会儿，因为我的餐品还没有完成。这一次呢，我给服务员留了一个电话，当餐备齐之后，让他给我打电话。这个中间呢，我可以出去干其他事情，不用一直在后厨等，也不用时不时的去问一下，这就是异步过程。




我们看一下上面这张图：当应用进程使用 AIO 函数从内核获取数据，这个时候内核中的数据还没有准备好，但是 AIO 直接返回了，剩下的工作全部由内核进行完成。在这个过程中，应用进程可以做自己的事情。

## 信号驱动

> Tips：这个形式的 I/O 很少使用，在这里我就不多赘述了，大家不必过多关注。

**总结**

这一部分的内容是理解 Nginx 事件机制的核心，Nginx 使用了 epoll，实现了 I/O 多路复用。

大家不必被网上的同步阻塞、同步非阻塞、异步阻塞、异步非阻塞的概念搞懵，这几个概念都是带有混淆性的，在《Unix 网络编程》卷一中，根本没有这几个概念，只有我们上面介绍的几个概念。

}

04 与新朋友 Nginx 友好交流
06 我快我有理：深扒 Nginx 工作原理