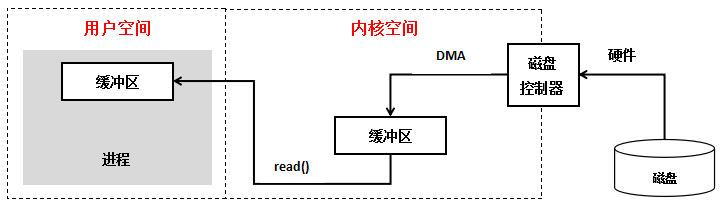
**IO模型**

　　linux系统IO分为内核准备数据和将数据从内核拷贝到用户空间两个阶段。



这张图大致描述了数据从外部磁盘向运行中程序的内存中移动的过程。

**用户空间、内核空间**

　　现在操作系统都是采用虚拟存储器，那么对32位操作系统而言，它的寻址空间（虚拟储存空间）为4G（2的32次方）。操作系统的核心是内核，独 立于普通的应用程序，可以访问受保护的内存空间，也有访问底层硬件设备的所有权限。为了保证用户进程不能直接操作内核，保证内核的安全，操作系统将虚拟空 间划分为两个部分，一个部分为内核空间，一部分为用户空间。

　　如何分配这两个空间的大小也是有讲究的，如windows 32位操作系统，默认的用户空间：内核空间的比例是1:1;而在32位Linux系统中的默认比例是3:1（3G用户空间，1G内核空间）。

**进程切换**

　　为了控制进程的执行，内核必须要有能力挂起正在CPU上运行的进程，并恢复以前挂起的某个进程的执行。这种行为成为进程的切换。任何进程都是在操作系统内核的支持下运行的，是与内核紧密相关的。

进程切换的过程，会经过下面这些变化：

1、保存处理机上下文，包括程序计数器和其他寄存器。

2、更新PCB信息。

3、将进程的PCB移入相应的队列，如就绪、在某事件阻塞等队列。

4、选择另外一个进程执行，并更新PCB

5、更新内存管理的数据结构。

6、恢复处理机上下文

**缓存IO**

　　缓存IO又称称为标准IO，大多数文件系统的默认IO操作都是缓存IO。在Linux的缓存IO机制中，操作系统会将IO的数据缓存在文件系统 的页缓存（page cache）。也就是说，数据会先被拷贝到操作系统内核的缓冲区中，然后才会从操作系统内核的缓存区拷贝到应用程序的地址空间中。

　　这种做法的缺点就是，需要在应用程序地址空间和内核进行多次拷贝，这些拷贝动作所带来的CPU以及内存开销是非常大的。

**同步、异步、阻塞、非阻塞**

**同步与异步**：描述的是用户线程与内核的交互方式，同步指用户线程发起IO请求后需要等待或者轮询内核IO操作完成后才能继续执行；而异步是指用户线程发起IO请求后仍然继续执行，当内核IO操作完成后会通知用户线程，或者调用用户线程注册的回调函数。

**阻塞与非阻塞**：描述是用户线程调用内核IO操作的方式，阻塞是指IO操作需要彻底完成后才返回到用户空间；而非阻塞是指IO操作被调用后立即返回给用户一个状态值，无需等到IO操作彻底完成。

**Linux IO模型**

　　网络IO的本质就是socket的读取，socket在linux系统被抽象为流，IO可以理解为对流的操作。文章开始的时候也提到了，对于一 次IO访问（以read为例），数据会先被拷贝到操作系统内核的缓冲区，然后才会从操作系统内核的缓冲区拷贝到应用程序的地址空间中。所以说，当一个 read操作发生时，它会经历两个阶段：

第一个阶段：等待数据准备。

第二个阶段：将数据从内核拷贝到进程中

对于socket流而言：

第一步：通常涉及等待网络上的数据分组到达，然后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。

当然，如果内核空间的缓冲区中已经有数据了，那么就可以省略第一步。至于为什么不能直接让磁盘控制器把数据送到应用程序的地址空间中呢？最简单的一个原因就是应用程序不能直接操作底层硬件。

网络应用需要处理的无非就是两大类问题，网络IO，数据计算。相对于后者，网络IO的延迟，给应用带来的性能瓶颈大于后者。网络IO的模型大致分为如下五种：

1、阻塞IO

2、非阻塞IO

3、多路复用IO

4、信号驱动IO

5、异步IO

前四种都是同步，只有最后一种是异步IO。下面的模型介绍先以生活中的例子来说明概念：周末和女友去商场逛街，到了晚上饭点，准备吃完饭再去逛街，但是周末人多，新白鹿饭店需要排队，于是有如下几种方案可供选择：

**1、阻塞IO模型**

**场景描述：**

　　在饭店领完号后，前面还有n桌，不知道什么时候到我们，但是又不能离开，因为过号之后必须重新取号。只好在饭店里等，一直等到叫号到我们才吃完晚饭，然后去逛街。中间等待的时间什么事情都不能做。

**网络模型：**

　　在这个模型中，应用程序为了执行这个read操作，会调用相应的一个system call，将系统控制权交给内核，然后就进行等待（这个等待的过程就是被阻塞了），内核开始执行这个system call，执行完毕后会向应用程序返回响应，应用程序得到响应后，就不再阻塞，并进行后面的工作。

优点：

　　能够及时返回数据，无延迟。

缺点：

　　对用户来说处于等待就要付出性能代价。

**2、非阻塞IO**

**场景描述：**

　　等待过程是在太无聊，于是我们就去逛商场，每隔一段时间就回来询问服务员，叫号是否到我们了，整个过程来来回回好多次。这就是非阻塞，但是需要不断的询问。

**网络模型：**

当用户进程发出read操作时，调用相应的system call，这个system call会立即从内核中返回。但是在返回的这个时间点，内核中的数据可能还没有准备好，也就是说内核只是很快就返回了system call，只有这样才不会阻塞用户进程，对于应用程序，虽然这个IO操作很快就返回了，但是它并不知道这个IO操作是否真的成功了，为了知道IO操作是否 成功，应用程序需要主动的循环去问内核。

优点：

　　能够在等待的时间里去做其他的事情。

缺点：

　　任务完成的响应延迟增大了，因为每过一段时间去轮询一次read操作，而任务可能在两次轮询之间的任意时间完成，这对导致整体数据吞吐量的降低。

**3、IO多路复用**

**场景描述：**

　　与第二个经常类似，饭店安装了电子屏幕，显示叫号的状态，所以在逛街的时候，就不用去询问服务员，而是看下大屏幕就可以了。（不仅仅是我们不用询问服务员，其他所有的人都可以不用询问服务员）

**网络模型：**

　　和第二种一样，调用system call之后，并不等待内核的返回结果而是立即返回。虽然返回结果的调用函数是一个异步的方式，但应用程序会被像select、poll和epoll等具 有多个文件描述符的函数阻塞住，一直等到这个system call有结果返回了，再通知应用程序。这种情况，从IO操作的实际效果来看，异步阻塞IO和第一种同步阻塞IO是一样的，应用程序都是一直等到IO操作 成功之后（数据已经被写入或者读取），才开始进行下面的工作。不同点在于异步阻塞IO用一个select函数可以为多个文件描述符提供通知，提供了并发 性。举个例子：例如有一万个并发的read请求，但是网络上仍然没有数据，此时这一万个read会同时各自阻塞，现在用select、poll、 epoll这样的函数来专门负责阻塞同时监听这一万个请求的状态，一旦有数据到达了就负责通知，这样就将一万个等待和阻塞转化为一个专门的函数来负责与管 理。

　　多路复用技术应用于JAVA NIO的核心类库多路复用器Selector中，目前支持I/O多路复用的系统调用有select、pselect、poll、epoll，在linux 编程中有一段时间一直在使用select做轮询和网络事件通知的，但是select支持一个进程打开的socket描述符（FD）收到了限制，一般为 1024，由于这一限制，现在使用了epoll代替了select，而epoll支持一个进程打开的FD不受限制。

　　异步IO与同步IO的区别在于：同步IO是需要应用程序主动地循环去询问是否有数据，而异步IO是通过像select等IO多路复用函数来同时检测多个事件句柄来告知应用程序是否有数据。

**了解了前面三种IO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，处理的方式是不一样的，直接等待、轮询、select或poll轮询，两个阶段过程：**

第一个阶段有的阻塞，有的不阻塞，有的可以阻塞又可以不阻塞。

第二个阶段都是阻塞的。

**从整个IO过程来看，他们都是顺序执行的，因此可以归为同步模型，都是进程自动等待且向内核检查状态。**

**高并发的程序一般使用同步非阻塞模式，而不是多线程+同步阻塞模式。**要 理解这点，先弄明白并发和并行的区别：比如去某部门办事需要依次去几个窗口，办事大厅的人数就是并发数，而窗口的个数就是并行度。就是说并发是同时进行的 任务数（如同时服务的http请求），而并行数就是可以同时工作的物理资源数量（如cpu核数）。通过合理调度任务的不同阶段，并发数可以远远大于并行 度。这就是区区几个CPU可以支撑上万个用户并发请求的原因。在这种高并发的情况下，为每个用户请求创建一个进程或者线程的开销非常大。而同步非阻塞方式 可以把多个IO请求丢到后台去，这样一个CPU就可以服务大量的并发IO请求。

　　IO多路复用究竟是同步阻塞还是异步阻塞模型，这里来展开说说：

　　同步是需要主动等待消息通知，而异步则是被动接受消息通知，通过回调、通知、状态等方式来被动获取消息。IO多路复用在阻塞到select阶段时，用户进程是主动等待并调用select函数来获取就绪状态消息，并且其进程状态为阻塞。所以IO多路复用是同步阻塞模式。

**4、信号驱动式IO**

　　应用程序提交 read请求，调用system call，然后内核开始处理相应的IO操作，而同时，应用程序并不等内核返回响应，就会开始执行其他的处理操作（应用程序没有被IO阻塞），当内核执行完 毕，返回read响应，就会产生一个信号或执行一个基于线程的回调函数来完成这次IO处理过程。在这里IO的读写操作是在IO事件发生之后由应用程序来完 成。异步IO读写操作总是立即返回，而不论IO是否阻塞，因为真正的读写操作已经有内核掌管。也就是说同步IO模型要求用户代码自行执行IO操作（将数据 从内核缓冲区移动用户缓冲区或者相反），而异步操作机制则是由内核来执行IO操作（将数据从内核缓冲区移动用户缓冲区或者相反）。可以这样认为，同步IO 向应用程序通知的是IO就绪事件，而异步IO向应用程序通知的是IO完成事件。

**5、异步IO**

　　异步IO与上面 的异步概念是一样的， 当一个异步过程调用发出后，调用者不能立刻得到结果，实际处理这个调用的函数在完成后，通过状态、通知和回调来通知调用者的输入输出操作。异步IO的工作 机制是：告知内核启动某个操作，并让内核在整个操作完成后通知我们，这种模型与信号驱动的IO区别在于，信号驱动IO是由内核通知我们何时可以启动一个 IO操作，这个IO操作由用户自定义的信号函数来实现，而异步IO模型是由内核告知我们IO操作何时完成。