漫话编程
2018年10月08日阅读 6050

漫话：如何给女朋友解释什么是Linux的五种IO模型?

周日午后，刚刚放下手里的电话，正在给刚刚的面试者写评价。刚刚写到『对Linux的基本IO模型理解不深』这句的时候，女朋友突然出现。

在Java中，主要有三种IO模型，分别是阻塞IO（BIO）、非阻塞IO（NIO）和 异步IO（AIO）。

Java中提供的IO有关的API，在文件处理的时候，其实依赖操作系统层面的IO操作实现的。比如在Linux 2.6以后，Java中NIO和AIO都是通过epoll来实现的，而在Windows上，AIO是通过IOCP来实现的。

可以把Java中的BIO、NIO和AIO理解为是Java语言对操作系统的各种IO模型的封装。程序员在使用这些API的时候，不需要关心操作系统层面的知识，也不需要根据不同操作系统编写不同的代码。只需要使用Java的API就可以了。

在Linux(UNIX)操作系统中，共有五种IO模型，分别是：阻塞**IO**模型、非阻塞**IO**模型、**IO**复用模型、信号驱动**IO**模型以及异步**IO**模型。

既然提到晚上吃鱼，那就通过钓鱼的例子来解释这五种IO模型吧。

到底什么是IO

我们常说的IO，指的是文件的输入和输出，但是在操作系统层面是如何定义IO的呢？到底什么样的过程可以叫做是一次IO呢?

拿一次磁盘文件读取为例，我们要读取的文件是存储在磁盘上的，我们的目的是把它读取到内存中。可以把这个步骤简化成把数据从硬件（硬盘）中读取到用户空间中。

其实真正的文件读取还涉及到缓存等细节，这里就不展开讲述了。关于用户空间、内核空间以及硬件等的关系如果读者不理解的话，可以通过钓鱼的例子理解。

钓鱼的时候，刚开始鱼是在鱼塘里面的，我们的钓鱼动作的最终结束标志是鱼从鱼塘中被我们钓上来，放入鱼篓中。

这里面的鱼塘就可以映射成磁盘，中间过渡的鱼钩可以映射成内核空间，最终放鱼的鱼篓可以映射成用户空间。一次完整的钓鱼（IO）操作，是鱼（文件）从鱼塘（硬盘）中转移（拷贝）到鱼篓（用户空间）的过程。

阻塞IO模型

我们钓鱼的时候，有一种方式比较惬意，比较轻松，那就是我们坐在鱼竿面前，这个过程中我们什么也不做，双手一直把着鱼竿，就静静的等着鱼儿咬钩。一旦手上感受到鱼的力道，就把鱼钓起来放入鱼篓中。然后再钓下一条鱼。

映射到Linux操作系统中，这就是一种最简单的IO模型，即阻塞IO。 阻塞 I/O 是最简单的 I/O 模型，一般表现为进程或线程等待某个条件，如果条件不满足，则一直等下去。条件满足，则进行下一步操作。

应用进程通过系统调用 `recvfrom` 接收数据，但由于内核还未准备好数据报，应用进程就会阻塞住，直到内核准备好数据报，`recvfrom` 完成数据报复制工作，应用进程才能结束阻塞状态。

这种钓鱼方式相对来说比较简单，对于钓鱼的人来说，不需要什么特制的鱼竿，拿一根够长的木棍就可以悠闲的开始钓鱼了（实现简单）。缺点就是比较耗费时间，比较适合那种对鱼的需求量小的情况（并发低，时效性要求低）。

非阻塞IO模型

我们钓鱼的时候，在等待鱼儿咬钩的过程中，我们可以做点别的事情，比如玩一把王者荣耀、看一集《延禧攻略》等等。但是，我们要时不时的去看一下鱼竿，一旦发现有鱼儿上钩了，就把鱼钓上来。

映射到Linux操作系统中，这就是非阻塞的IO模型。应用进程与内核交互，目的未达到之前，不再一味的等着，而是直接返回。然后通过轮询的方式，不停的去问内核数据准备有没有准备好。如果某一次轮询发现数据已经准备好了，那就把数据拷贝到用户空间中。

应用进程通过 recvfrom 调用不停的去和内核交互，直到内核准备好数据。如果没有准备好，内核会返回error，应用进程在得到error后，过一段时间再发送recvfrom请求。在两次发送请求的时间段，进程可以先做别的事情。

这种方式钓鱼，和阻塞IO比，所使用的工具没有什么变化，但是钓鱼的时候可以做些其他事情，增加时间的利用率。

信号驱动IO模型

我们钓鱼的时候，为了避免自己一遍一遍的去查看鱼竿，我们可以给鱼竿安装一个报警器。当有鱼儿咬钩的时候立刻报警。然后我们再收到报警后，去把鱼钓起来。

映射到Linux操作系统中，这就是信号驱动IO。应用进程在读取文件时通知内核，如果某个 socket 的某个事件发生时，请向我发一个信号。在收到信号后，信号对应的处理函数会进行后续处理。

应用进程预先向内核注册一个信号处理函数，然后用户进程返回，并且不阻塞，当内核数据准备就绪时会发送一个信号给进程，用户进程便在信号处理函数中开始把数据拷贝的用户空间中。

这种方式钓鱼，和前几种相比，所使用的工具有了一些变化，需要有一些定制（实现复杂）。但是钓鱼的人就可以在鱼儿咬钩之前彻底做别的事儿去了。等着报警器响就行了。

IO复用模型

我们钓鱼的时候，为了保证可以最短的时间钓到最多的鱼，我们同一时间摆放多个鱼竿，同时钓鱼。然后哪个鱼竿有鱼儿咬钩了，我们就把哪个鱼竿上面的鱼钓起来。

映射到Linux操作系统中，这就是IO复用模型。多个进程的IO可以注册到同一个管道上，这个管道会统一和内核进行交互。当管道中的某一个请求需要的数据准备好之后，进程再把对应的数据拷贝到用户空间中。

IO多路转接是多了一个`select`函数，多个进程的IO可以注册到同一个`select`上，当用户进程调用该`select`，`select`会监听所有注册好的IO，如果所有被监听的IO需要的数据都没有准备好时，`select`调用进程会阻塞。当任意一个IO所需的数据准备好之后，`select`调用就会返回，然后进程在通过`recvfrom`来进行数据拷贝。

这里的IO复用模型，并没有向内核注册信号处理函数，所以，他并不是非阻塞的。进程在发出`select`后，要等到`select`监听的所有IO操作中至少有一个需要的数据准备好，才会有返回，并且也需要再次发送请求去进行文件的拷贝。

这种方式的钓鱼，通过增加鱼竿的方式，可以有效的提升效率。

为什么以上四种都是同步的

我们说阻塞IO模型、非阻塞IO模型、IO复用模型和信号驱动IO模型都是同步的IO模型。原因是因为，无论以上那种模型，真正的数据拷贝过程，都是同步进行的。

信号驱动难道不是异步的么？ 信号驱动，内核是在数据准备好之后通知进程，然后进程再通过`recvfrom`操作进行数据拷贝。我们可以认为数据准备阶段是异步的，但是，数据拷贝操作是同步的。所以，整个IO过程也不能认为是异步的。

我们把钓鱼过程，可以拆分为两个步骤：1、鱼咬钩（数据准备）。2、把鱼钓起来放进鱼篓里（数据拷贝）。无论以上提到的哪种钓鱼方式，在第二步，都是需要人主动去做的，并不是鱼竿自己完成的。所以，这个钓鱼过程其实还是同步进行的。

烧水的报警器一响，整个烧水过程就完成了。水已经是开水了。

钓鱼的报警器一响，只能说明鱼儿已经咬钩了，但是还没有真正的钓上来。

所以，使用带有报警器的水壶烧水，烧水过程是异步的。

而使用带有报警器的鱼竿钓鱼，钓鱼的过程还是同步的。

异步IO模型

我们钓鱼的时候，采用一种高科技钓鱼竿，即全自动钓鱼竿。可以自动感应鱼上钩，自动收竿，更厉害的可以自动把鱼放进鱼篓里。然后，通知我们鱼已经钓到了，他就继续去钓下一条鱼去了。

映射到Linux操作系统中，这就是异步IO模型。应用进程把IO请求传给内核后，完全由内核去操作文件拷贝。内核完成相关操作后，会发信号告诉应用进程本次IO已经完成。

用户进程发起aio_read操作之后，给内核传递描述符、缓冲区指针、缓冲区大小等，告诉内核当整个操作完成时，如何通知进程，然后就立刻去做其他事情了。当内核收到aio_read后，会立刻返回，然后内核开始等待数据准备，数据准备好以后，直接把数据拷贝到用户控件，然后再通知进程本次IO已经完成。

这种方式的钓鱼，无疑是最省事儿的。啥都不需要管，只需要交给鱼竿就可以了。

5种IO模型对比

介绍完这些之后，我默默的删掉了之前写好的那句面试评价『对Linux的基本IO模型理解不深』，改成了『对IO体系理解的不够深入，只会使用封装好的API』。