其实 Linux IO 模型没那么难

Posted by 陈树义 on 2021-06-30

IO 其实就是 Input 和 Output，在操作系统中就对应数据流的输入与输出。这个数据流的两端，可以是文件，也可以是网络的一台主机。但无论是文件，还是网络主机，其传输都是类似的，我们今天就以源头为文件进行说明。

一个文件要从磁盘到我们的内存，需要经过很复杂的操作。首先，需要将数据从硬件读取出来，然后放入操作系统内核缓冲区，之后再将数据拷贝到程序缓冲区，最后应用程序才能读取到这个文件。简单地说，无论什么 IO 模型，其读取过程总会经历下面两个阶段：

- 等待数据到达内核缓冲区
- 从内核缓冲区拷贝数据到程序缓冲区

而我们 Linux 根据这两个阶段的是否阻塞，分成了 5 个经典的 IO 的模型，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
- 异步 IO 模型

阻塞 IO 模型

阻塞 IO 称为 Blocking IO，简称 BIO。在阻塞 IO 模型中，当进程发起一个读取文件请求（recvfrom 系统调用）时，如果内核缓存区没有对应的数据，那么它不会立刻恢复，而是去读取磁盘数据，当数据读取完毕后，再返回给进程。此时，第一个阶段完成。在这个阶段进程是阻塞的，因为它要等待内核将数据读取到内核缓冲区。

而当进程收到内核的响应之后，进程再把数据从内核缓冲区复制到程序缓冲区，最后完成文件读取操作。此时，第二个阶段完成。在这个阶段进程也是阻塞的，因为它要将数据从内核缓冲区拷贝到程序缓冲区。

简单地说：在阻塞 IO 模型里，从硬件到系统内核、从系统内核到程序空间，都是阻塞的。

非阻塞 IO 模型

在非阻塞 IO 模型下，当一个请求发起读取文件请求（recvfrom）时，如果内核缓冲区没有数据，那么内核会读取文件数据。但此时请求并不会阻塞，而是返回一个错误信息（EWOULDBLOCK）告诉进程：数据暂时还没准备好，你待会儿再试试。

于是进程就不断地向内核重试，问：数据准备好了没有，数据准备好了没有……当内核准备好数据，进程就会收到对应消息，于是第一阶段就结束了。非阻塞 IO 中的非阻塞说的就是进程不会阻塞在这里，而是会不断重试。

虽然说这样并没有太大用处，反而会使得 CPU 空转，但总比之前有了一点进步。在这个阶段进程并不是阻塞的。当进程得知内核准备好数据之后，其便会将数据从内核缓冲区拷贝到程序缓冲区。这个阶段与阻塞 I/O 模型是完全一样的，同样是会导致进程阻塞。

简单地说：在非阻塞 IO 模型里，从硬件到系统内核、从系统内核到程序空间，同样都是阻塞的。但是其比阻塞 IO 争气了一点，并不是站在那里不动，好歹还跑了一下。虽然是在做无用功，但是好歹提高了一丢丢效率。

IO 复用模型

IO 复用之所以叫复用，是因为其能同时操作多个数据流。而前面的 阻塞 IO、非阻塞 IO 同一时间只能操作一个数据流。在 IO 复用模型中，进程监听多个数据流并阻塞，当任何一个数据流有数据之后，其便会收到内核的响应。此时，第一个阶段完成，在这个阶段进程其实是阻塞的。

而当收到内核的响应后，进程便会将数据从内核缓冲区复制到程序缓冲区。这个阶段与上面两个模型一模一样，进程同样阻塞。

简单地说：IO 复用模型在第二阶段与阻塞 IO 和非阻塞 IO 是完全一致的。但是在第一阶段上，其有效率上的巨大提升，其能同时轮询多个数据流，提高了效率。

信号驱动 IO 模型

信号驱动与前面几个模型的不同之处就在与信号这个词。信号驱动 IO 在第一阶段，即数据到达内核缓冲区之前，进程是不阻塞的，而是设置一个信号回调。当数据到达内核缓冲区之后，内核调用程序的回调。通过这种方式，信号驱动 IO 下的进程就可以不阻塞，可以去做其他事情了。

而当进程收到信号，进程再将数据从内核缓冲区复制到程序缓冲区。这个过程与上面几个是完全一样的，同样也是阻塞的。

信号驱动 IO 可以说是 IO 读取的一个里程碑，其真正实现了异步读取数据。信号驱动 IO 其二个阶段，与上面几个是一样的。但是其在第一个阶段做到了真正的异步。信号驱动 IO 在第一阶段，其去请求内核读取数据，这时候其不会阻塞，也不会去寻轮，而是设置一个信号回调。 当数据完全拷贝到系统内核时，系统发出 SIGIO 信号，通知进程去进行第二阶段，将数据拷贝到程序缓冲区。

异步 IO 模型

异步 IO 相比前面几个流程，真正做到了完全非阻塞。无论是在第一阶段，还是在第二阶段都是非阻塞。与信号驱动 IO 类似，异步 IO 模型通过信号回调的方式，在第一个阶段实现了进程的非阻塞。而当数据到达内核缓冲区之后，进程便会收到通知。

而当进程收到通知之后，进程再次将数据从内核缓冲区复制到进程缓冲区，但这时进程并不等待，而是同样设置一个信号回调。当复制完成后，进程收到通知，再进行相应的处理。

异步 IO 与信号驱动 IO 相比，做得更加彻底了！

异步 IO 不仅仅是在第一阶段实现了信号回调，其也在第二阶段实现了信号回调，从而完全实现了异步 IO 操作。

总结

我们回顾一下这 5 种 IO 模型：

- 阻塞 IO 模型：硬件到系统内核，阻塞。系统内核到程序空间，阻塞。
- 非阻塞 IO 模型：硬件到系统内核，轮询阻塞。系统内核到程序空间，阻塞。
- 复用 IO 模型：硬件到系统内核，多流轮询阻塞。系统内核到程序空间，阻塞。
- 信号驱动 IO 模型：硬件到系统内核，信号回调不阻塞。系统内核到程序空间，阻塞。
- 异步 IO 模型：硬件到系统内核，信号回调不阻塞。系统内核到程序空间，信号回调不阻塞。

从上面的 5 种 IO 模型，我们可以看出，真正实现异步非阻塞的只有异步 IO 这种模型，而其他四种都是同步性 IO。因为在第二阶段：从内核缓冲区复制到进程缓冲区的时候，不可能干其他事情。

好了，关于 Linux IO 模型的分享，今天就聊到这儿。

谢谢大家的阅读。如果文章对你有帮助，欢迎评论转发点赞三连，我们下次见~

- 上一篇
- 下一篇

目录

- 阻塞 IO 模型
- 非阻塞 IO 模型
- IO 复用模型
- 信号驱动 IO 模型
- 异步 IO 模型
- 总结

FEATURED TAGS

性能优化 单测 事务 Spring 性能调优 Tomcat MySQL 系统设计 稳定性建设 synchronized 并发编程 Java内存模型 思维误区 认知成长 简历 爬虫 Github 邮件 经济学 书籍推荐 年度总结 个税 排序 算法 程序员 架构师 软件工程 操作系统 阻塞队列源码系列 推送基础系列 JVM 规范系列 Prometheus 入门系列 集合源码系列 JVM 基础系列 并发集合源码系列 并发包源码系列 线程池源码系列 JVM实战 Apache Common Pool 树结构 数据结构 中年危机 教员 Redis HBase 有赞 Chrome 技术管理 美团 建站 Kafka 法律 Prometheus 商业 哲学 时间管理 Markdown 面试 华为 Maven 区块链 源码 雷军 小米 线上问题 管理 方法论 数据库 Push JVM Alfred 架构设计 计算机原理 MongoDb 职业规划 运维 重构 设计模式 LOG4J ImageMagick 计算机网络 入门教程 毛主席 Java Canal ElasticSearch Linux Shell

---

FRIENDS

- 田小波的博客
- *知*