Java通过零拷贝实现高效的数据传输

大哥你先走 关注

2 2019.01.17 20:51:48 字数 2,625 阅读 3,148

零拷贝，零开销

> 本文仅是中文版本，原文由 Sathish Palaniappan, Pramod Nagaraja 发布于 2008年09月2号。文章适合初次接触零拷贝技术并想进一步学习的读者，零拷贝本身是一种思想，不与任何编程语言绑定，不懂Java的读者可以跳过零拷贝技术在Java中实现的具体细节。

许多Web应用提供大量的静态内容，主要就是从磁盘读取数据然后将数据写回套接字，中间不涉及数据的变换。这种操作对CPU的使用相对较少，但是效率很低：首先，内核从文件读取数据，然后将数据从内核空间拷贝到用户进程空间，最后应用程序将数据拷贝回内核空间并通过套接字发送。实际上，在整个流程中应用程序仅充当一个将数据从磁盘拷贝到套接字的低效中间层。

每次数据跨越用户态和内核态的边界，数据都需要拷贝，拷贝操作消耗CPU和内存带宽。幸运的是通过一种称为"零拷贝"的技术可消除这些不必要的拷贝。使用零拷贝的应用要求内核将磁盘数据直接拷贝到套接字而不再经过应用。零拷贝可以极大的提高应用的性能并减少上下文在内核态和用户态之间的切换次数。

在 Linux 和 Unix 系统中 Java 类库通过 `java.nio.channels.FileChannel` 的 `transgerTo` 方法支持零拷贝。可以使用 `transgerTo` 方法在两个通道之间直接传递数据，而不要求数据经过应用程序。为了更好的理解零拷贝技术对性能的提升，首先通过传统复制语义实现一个简单文件传输功能，然后通过零拷贝技术实现同样功能，并比较两种实现在性能上的差异。

数据传输: 传统语义

考虑这样的场景：从文件读取数据并通过网络将数据传递给其他程序（这是很多应用的行为，包括提供静态内容的Web应用，FTP 服务器，邮件服务器等）。两个核心的操作如代码1所示：

代码 1. 从文件拷贝数据到套接字

```
File.read(fileDesc, buf, len);
Socket.send(socket, buf, len);
```

虽然代码1非常的简单，但是在代码内部实现，拷贝操作需要上下文在用户态和内核态切换四次，在操作完成前数据需要拷贝四次。图1展示了数据如何从文件转移到套接字：

大哥你先走 关注
总资产97 (约9.83元)

学习笔记-final关键字
阅读 21

快速构建Spring Boot Starter指南
阅读 41

推荐阅读

Spring源码----Spring的Bean生命周期流程图及代码解释
阅读 2,659

蚂蚁二面，面试官问我零拷贝的实现原理，当场懵了...
阅读 14,496

一文搞定Java热更新
阅读 964

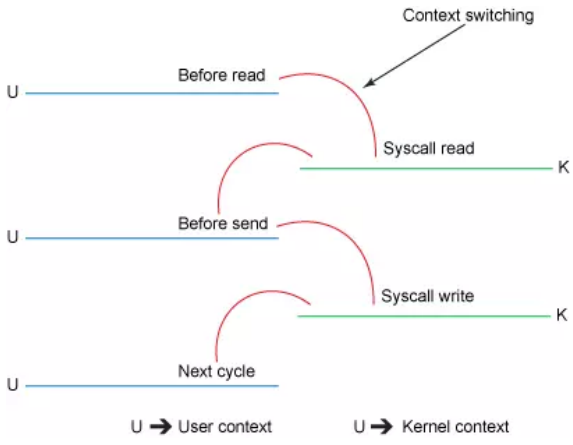
20000字Java面试题解析（事务+缓存+数据库+多线程+JVM）
阅读 1,815

优雅的去除ifelse代码
阅读 2,303

数据拷贝路径

图 1. 传统数据拷贝方法

图 2 展示了上下文切换:



上下文切换

图 2. 传统方法下的上下文切换

涉及的步骤包括:

1. `read()` 调用导致上下文从用户态切换到内核态。内核通过 `sys_read()` （或等价的方法）从文件读取数据。DMA引擎执行第一次拷贝：从文件读取数据并存储到内核空间的缓冲区。
2. 请求的数据从内核的读缓冲区拷贝到用户缓冲区，然后 `read()` 方法返回。 `read()` 方法返回导致上下文从内核态切换到用户态。现在待读取的数据已经存储在用户空间内的缓冲区。
3. `send()` 调用导致上下文从用户态切换到内核态。第三次拷贝数据从用户空间重新拷贝到内核空间缓冲区。但是，这一次，数据被写入一个不同的缓冲区，一个与目标套接字相关联的缓冲区。
4. `send()` 系统调用返回导致第四次上下文切换。当DMA引擎将数据从内核缓冲区传输到协议引擎缓冲区时，第四次拷贝是独立且异步的。

使用中间内核缓冲区（而不是将数据直接发送到用户缓冲区）似乎非常低效。但是，进程引入中间内核缓冲区可以提高性能。在读取端使用中间内核缓冲区，在应用请求的数据没有超出内核缓冲区的数据时，内核缓冲区可以担当"预读缓存"的角色。在写端，中间内核缓冲区使写操作完全异步化。

不幸的是，当请求的数据大于内核缓冲区大小时这种方法往往会成为性能瓶颈。数据在最终被发送之前，在磁盘，内核缓冲区和用户缓冲区之间发生多次拷贝。零拷贝通过减少不必要的数据拷贝以提供性能。

数据传输: 零拷贝方式

如果你回想使用传统语义传递数据的场景，你会发现第二次和第三次数据拷贝并不是真的需要。应用程序除了缓存数据然后将数据传回套接字缓冲区外没有做任何事情。数据可以直接从内核的读缓冲区传输到套接字缓冲区。 `transferTo` 方法允许你实现这样的流程。 `transferTo` 方法的签名如

Java通过零拷贝实现高效的数据传输

大哥你先走 关注 赞赏支持

```
public void transferTo(long position, long count, WritableByteChannel target);
```

transferTo 方法将数据从文件通道传输到给定的可写字节通道。 transferTo 内部实现依赖底层操作系统对零拷贝的支持：在UNIX和各 Linux 版本中， transgerTo 方法调用最终会调用 sendfile()方法，代码如 List 3 所示， sendfile 将数据从一个文件描述符传输到另一个：

代码 3. The sendfile() system call

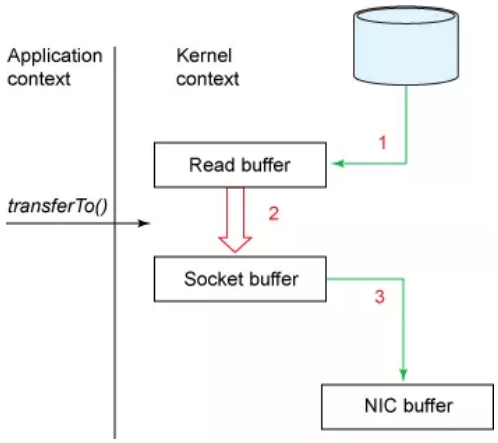
```
#include <sys/socket.h>
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

代码 1 中的 file.read() 和 socket.send() 两个方法调用可以替换为一个 transferTo() 方法调用，如代码 4所示：

代码 4. **使用** transferTo() **从磁盘拷贝数据到套接字**

```
transferTo(position, count, writableChannel);
```

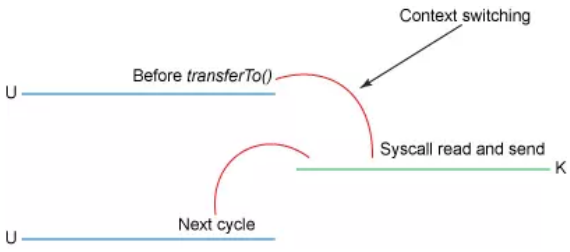
图 3 展示了使用 transferTo() 方法时，数据的流向:



数据拷贝路径

图 3. **使用**transferTo()**时数据拷贝**

图 4 展示了使用 transferTo() 方法时，上下文的切换：



上下文切换

图 4. **使用** transferTo() **时上下文切换**

使用 transgerTo() 方法时涉及的步骤包括以下两步：

写下你的评论... 评论1 赞16 ···

Java通过零拷贝实现高效的数据传输

大哥你先走 关注 赞赏支持

2. DMA引擎将数据从内核套接字缓冲区传输到协议引擎（第三次数据拷贝）。

这是一个改进：上下文切换的次数从4次减少到2次，数据拷贝的次数从4次减少到3次（仅有一次数据拷贝消耗CPU资源）。然而，这并没有实现零拷贝的目标，如果底层网卡支持*gather operations*，可以进一步减少内核拷贝数据的次数。Linux 内核 从2.4 版本开始修改了套接字缓冲区描述符以满足这个要求。这种方法不仅减少了多个上下文切换，还消除了消耗CPU的重复数据拷贝。用户使用的方法没有任何变化，依然通过 `transferTo` 方法，但是方法的内部实现

发生了变化：

1. `transferTo` 方法调用触发 DMA 引擎将文件上下文信息拷贝到内核缓冲区。

2. 数据不会被拷贝到套接字缓冲区，只有数据的描述符（包括数据位置和长度）被拷贝到套接字缓冲区。DMA 引擎直接将数据从内核缓冲区拷贝到协议引擎，这样减少了最后一次需要消耗CPU的拷贝操作。

图 5 展示了在有 `gather option` 条件下使用 `transferTo` 时数据拷贝情况：

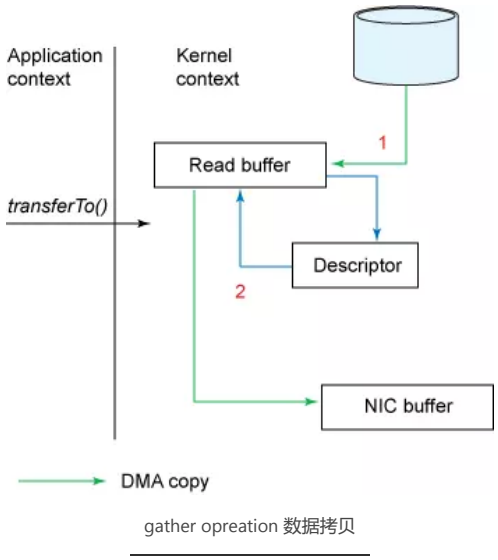


图 5. 使用 transferTo() and gather operations 时的数据拷贝

性能测试

现在让我们在一个需要在客户端和服务器之间传输文件的程序中应用零拷贝。`TraditionalClient.java` 和 `TraditionalServer.java` 基于传统复制语义，使用 `File.read()` 和 `Socket.send()` 方法读取和发送数据. `TraditionalServer.java` 是一个监听在5230端口等待客户端连接的服务器应用，每次从套接字中读取4kb的数据。 `TraditionalClient.java` 连接到服务器, 使用 `File.read()` 方法每次从文件读取4kb数据，然后调用方法 `socket.send()`) 将数据通过套接字发送给服务器.

类似的, `TransferToServer.java` 和 `TransferToClient.java` 通过使用 `transferTo()` （使用 `sendfile()` 系统调用发送数据）实现一样的将数据从客户端发送到服务器的功能。

性能比较

在Linux 内核2.6版本上，以毫秒统计使用传统方法和使用 `transferTo` 方法传输不同大小的文件的耗时。表1展示了测试结果：

Java通过零拷贝实现高效的数据传输

7MB	156	45
21MB	337	128
63MB	843	387
98MB	1320	617
200MB	2124	1150
350MB	3631	1762
700MB	13498	4422
1GB	18399	8537

从测试结果来看使用 `transgerTo` 的API和传统方法相比可以降低65%的传输时间。这可以有效的提高在不同I/O通道之间大量拷贝数据应用的性能。

总结

我们已经证明了在从一个通道读取数据并将相同的数据写入另一个通道的场景下使用 `transferTo` 带来的巨大性能优势。内部缓冲区的拷贝，尽管这些拷贝隐藏在内核里，但是也是可观的消耗。对于需要处理在通道之间拷贝大量数据的应用，零拷贝技术可以显著的提升性能。性能测试使用的用例可以从Github免费下载。

扩展阅读

在Java编程领域，Netty是一个非常流行的基于事件驱动的异步网络应用框架，Netty的核心框架之一就是拥有丰富的支持零拷贝的字节缓冲区，想进一步了解零拷贝技术的朋友可以深入研究Netty中零拷贝技术的实现。

Java通过零拷贝实现高效的数据传输

大哥你先走 关注 赞赏支持

transgerTo写错了。应该是transferTo😊

赞 回复

被以下专题收入，发现更多相似内容

＋收入我的专题 Amazing...

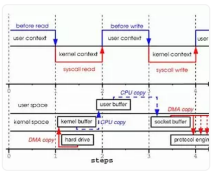
推荐阅读

更多精彩内容 >

浅谈 Linux下的零拷贝机制

什么是零拷贝 维基上是这么描述零拷贝的：零拷贝描述的是CPU不执行拷贝数据从一个存储区域到另一个存储区域的任务，这...

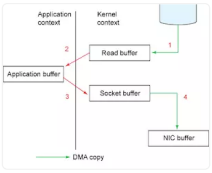
tomas家的小拨浪鼓 阅读 4,784 评论 9 赞 38



通过zero-copy进行高效的数据传输(译)

这篇文章翻译自Efficient data transfer through zero copy。由于译者水平有限...

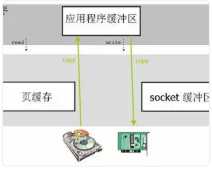
AlstonWilliams 阅读 191 评论 0 赞 1



Linux 中的零拷贝技术

【转自】：https://www.ibm.com/developerworks/cn/linux/l-cn-zer...

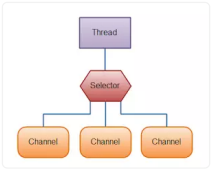
lxqfirst 阅读 277 评论 0 赞 5



Java NIO

Java IO 的底层原理 缓冲处理、内核空间与用户空间 缓冲与缓冲的处理方式,是所有I/O操作的基础。术语"输入...

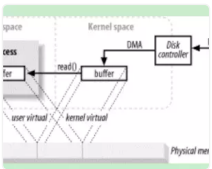
懒癌正患者 阅读 862 评论 1 赞 9



JAVA IO 以及 NIO 理解（转）

转自JAVA IO 以及 NIO 理解 一段话总结：传统io中从磁盘中中读文件，并把文件通过网络(socket)发...

抓兔子的猫 阅读 598 评论 0 赞 4



写下你的评论... 评论1 赞16 ...