Linux的高效的数据传输技术-Relay PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/145/2021_2022_Linux_E7_9A_84_E9_AB_c103_145253.htm Relay 是一种从 Linux 内核到用户空间的高效数据传输技术。通过用户定义的 relay 通道，内核空间的程序能够高效、可靠、便捷地将数据传输到用户空间。Relay 特别适用于内核空间有大量数据需要传输到用户空间的情形，目前已经广泛应用在内核调试工具如 SystemTap中。本文介绍了 Relay 的历史和原理，并且用一个简单的实例介绍了 Relay 的具体用法。 Relay 要解决的问题 对于任何在内核工作的程序而言，如何把大量的调试信息从内核空间传输到用户空间都是一个大麻烦，对于运行中的内核更是如此。特别是对于哪些用于调试内核性能的工具，更是如此。 对于这种大量数据需要在内核中缓存并传输到用户空间需求，很多传统的方法都已到达了极限，例如内核程序员很熟悉的 printk() 调用。此外，如果不同的内核子系统都开发自己的缓存和传输代码，造成很大的代码冗余，而且也带来维护上的困难。这些，都要求开发一套能够高效可靠地将数据从内核空间转发到用户空间的系统，而且这个系统应该独立于各个调试子系统。这样就诞生了 RelayFS。 Relay的发展历史 Relay 的前身是 RelayFS，即作为 Linux 的一个新型文件系统。2003年3月，RelayFS的第一个版本的代码被开发出来，在7月14日，第一个针对2.6内核的版本也开始提供下载。经过广泛的试用和改进，直到2005年9月，RelayFS才被加入mainline内核(2.6.14)。同时，RelayFS也被移植到2.4内核中。在 2006年2月，从2.6.17开始，RelayFS不再作为单独的文件系统存在，而是成为内核

的一部分。它的源码也从fs/目录下转移到 kernel/relay.c中，名称中也从RelayFS改成了Relay。 RelayFS目前已经被越来越多的内核工具使用，包括内核调试工具SystemTap、LTT，以及一些特殊的文件系统例如DebugFS。 Relay的基本原理 总的说来，Relay提供了一种机制，使得内核空间的程序能够通过用户定义的relay通道(channel)将大量数据高效的传输到用户空间。 一个relay通道由一组和CPU一一对应的内核缓冲区组成。这些缓冲区又被称为relay缓冲区(buffer)，其中的每一个在用户空间都用一个常规文件来表示，这被叫做relay文件(file)。内核空间的用户可以利用relay提供的API接口来写入数据，这些数据会被自动的写入当前的 CPU id对应的那个relay缓冲区；同时，这些缓冲区从用户空间看来，是一组普通文件，可以直接使用read()进行读取，也可以使用mmap()进行映射。Relay并不关心数据的格式和内容，这些完全依赖于使用relay的用户程序。Relay的目的是提供一个足够简单的接口，从而使得基本操作尽可能的高效。 Relay将数据的读和写分离，使得突发性大量数据写入的时候，不需要受限于用户空间相对较慢的读取速度，从而大大提高了效率。Relay作为写入和读取的桥梁，也就是将内核用户写入的数据缓存并转发给用户空间的程序。这种转发机制也正是Relay这个名称的由来。 这里的relay通道由四个relay缓冲区(kbuf0到kbuf3)组成，分别对应于系统中的cpu0到cpu1。每个CPU上的代码调用relay_write()的时候将数据写入自己对应的relay缓冲区内。每个relay缓冲区称一个relay文件，即/cpu0到 /cpu3。当文件系统被mount到/mnt/以后，这个relay文件就被映射成映射到用户空间的地址空间。一旦数据可用，用户程序就可以把它的

数据读出来写入到硬盘上的文件中，即cpu0.out到cpu3.out。Relay的主要API 前面提到的 relay_write() 就是 relay API 之一。除此以外，Relay 还提供了更多的 API来支持用户程序完整的使用 relay。这些 API，主要按照面向用户空间和面向内核空间分为两大类，下面我们来分别进行介绍。 面向用户空间的 API 这些 Relay 编程接口向用户空间程序提供了访问 relay 通道缓冲区数据的基本操作的入口，包括： ●open() - 允许用户打开一个已经存在的通道缓冲区。 ●mmap() - 使通道缓冲区被映射到位于用户空间的调用者的地址空间。要特别注意的是，我们不能仅对局部区域进行映射。也就是说，必须映射整个缓冲区文件，其大小是 CPU的个数和单个 CPU 缓冲区大小的乘积。 ●read() - 读取通道缓冲区的内容。这些数据一旦被读出，就意味着他们被用户空间的程序消费掉了，也就不能被之后的读操作看到。 ●sendfile() - 将数据从通道缓冲区传输到一个输出文件描述符。其中可能的填充字符会被自动去掉，不会被用户看到。 ●poll() - 支持 POLLIN/POLLRDNORM/POLLERR 信号。每次子缓冲区的边界被越过时，等待着的用户空间程序会得到通知。 ●close() - 将通道缓冲区的引用数减1。当引用数减为0时，表明没有进程或者内核用户需要打开它，从而这个通道缓冲区被释放。面向内核空间的 API 这些API接口向位于内核空间的用户提供了管理relay通道、数据写入等功能。下面介绍其中主要的部分，完整的API接口列表请参见这里。 ●relay_open() - 创建一个relay通道，包括创建每个CPU对应的relay缓冲区。 ●relay_close() - 关闭一个relay通道，包括释放所有的relay缓冲区，在此之前会调用relay_switch()来处理这些relay缓冲区以保

证已读取但是未满的数据不会丢失 ●relay_write() - 将数据写入到当前CPU对应的relay缓冲区内。由于它使用了local_irqsave()保护，因此也可以在中断上下文中使用。 ●relay_reserve() - 在relay通道中保留一块连续的区域来留给未来的写入操作。这通常用于那些希望直接写入到relay缓冲区的用户。考虑到性能或者其它因素，这些用户不希望先把数据写到一个临时缓冲区中，然后再通过relay_write()进行写入。 