第 32 卷 第 15 期 计 算 机 工 程 2006 年 8 月
Vol.32 № 15 Computer Engineering August 2006
·开发研究与设计技术· 文章编号：1000—3428(2006)15—0271—03 文献标识码：A 中图分类号：TP393.41

Linux 下实时图像传输卡驱动程序的研究与实现

王岳环，路自谦
(华中科技大学图像识别与人工智能研究所图像信息处理与智能控制国家教育部重点实验室，武汉 430074)

摘 要：实时图像传输对许多计算机视觉和图像处理应用都很重要，图像传输卡的实时性在很大程度上取决于驱动程序的高效实现。文章介绍了 Linux 下驱动程序的体系结构，研究并实现了一种 PCI 接口的实时图像传输卡在 Linux 下的驱动程序，通过实验分析了驱动程序中的一些关键参数对传输性能的影响，并对传输卡在 Linux 和 Windows 下的传输速率进行了比较。
关键词：Linux；实时 PCI；驱动程序

Research and Implementation of Device Driver for a Real-time Image Transmission Card Under Linux

WANG Yuehuan, LU Ziqian
(Key Lab for Image Processing and Intelligent Control, Ministry of Education, Institute for Pattern Recognition and Artificial Intelligence, Huazhong University of Science and Technology, Wuhan 430074)

【Abstract】 Real-time image transmission is a key discipline for a lot of applications like computer vision and image processing. This paper introduces the Linux device driver architecture, followed by a detailed analysis of the device driver for a PCI-interfaced real-time image transmission card under Linux. The analysis focuses on some key parameters that influence the transmission speed most. The speed of the card under Linux and Windows are compared.
【Key words】 Linux; Real-time PCI; Device-driver

传输卡是图像传输系统中一个很重要的设备，而驱动程序是硬件和操作系统之间的桥梁，对硬件设备的性能有很大影响，高效实现驱动程序才能将来自PC的图像数据高速实时地发送给图像处理模块。本文针对实时图像传输卡在Linux系统下驱动程序的高效实现进行了研究。由于 32 位的PCI总线传输速率可以高达 133Mbps[1]，因此传输卡采用PCI接口。Linux具有多种优势，如完全公开源代码、具有很强的稳定性、嵌入性和模块性等，因此这里采用Linux作为此传输卡的主机操作系统。

1 Linux 驱动程序的体系结构

1.1 Linux 驱动程序体系结构的起源

Linux是Unix操作系统的一个克隆，最初由Linus Torvalds创造[2]。它和Unix有着类似的体系结构。Unix操作系统将设备视为文件系统的节点，一般将所有的设备文件节点放到同一个知名目录下[3]。将设备表示成文件系统节点的目的是允许应用程序以一种设备无关的方式访问设备。应用程序仍然可以使用设备I/O控制操作对设备进行设备相关的访问。设备由它的主设备号和从设备号标识。主设备号作为索引值索引一个由驱动程序构成的数组，从设备号用于将相似的物理设备归为一组[3]。Unix系统下存在两类设备：字符型和块型。字符型设备驱动允许对设备进行无缓冲的顺序访问；块设备的驱动允许对设备随机存储，并且使用了缓冲。块设备必须作为文件系统节点被加载才能被访问[4]。

1.2 Linux 驱动程序体系结构

Linux下的驱动由模块表示，模块扩展了Linux内核的功能。Linux的驱动框架如图 1 所示。模块在加载的时候输出所有它提供给其他模块和内核调用的符号表，此表由内核维护，模块之间的通信就通过这些功能调用完成[5]。

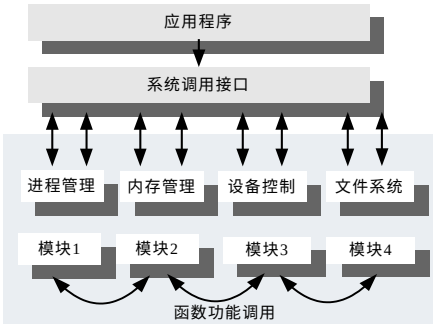

图 1 Linux 驱动程序的框架

2 Linux 系统下实时图像传输卡的驱动程序实现

在 Linux 系统下，将图像传输卡当作字符型设备来处理，对待处理的图像数据使用 DMA 将其从 PC 通过传输卡传输到目的地。DMA 传输完成后会产生中断，在中断服务例程和延迟过程调用中通过设置事件通知应用程序一次 DMA 传输已经结束，应用程序可以开始第 2 次 DMA 传输，如此不停地继续下去，直到把所有的图像数据传输完。图 2 是传输卡的数据流。

基金项目：国家自然科学基金重点资助项目(60135020)；国防预研基金资助项目
作者简介：王岳环(1972—)，男，博士，主研方向：图像分析，计算机主动视觉，精确制导技术，高性能 DSP 系统，智能仪器等；路自谦，硕士
收稿日期：2006-03-23 **E-mail：**ziqian.lu@gmail.com

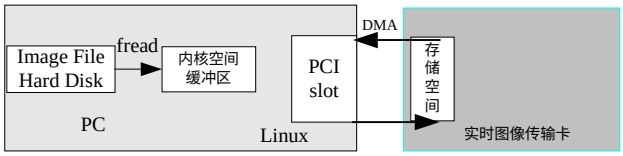

图 2 传输卡的数据流

下面详细介绍在 Linux 系统下传输卡驱动程序编写的主要步骤及其难点和解决方法。

2.1 内核空间缓冲区的读写

在传输图像数据之前，首先要对用于 DMA 传输的系统内存加载数据，即将一帧图像数据读入到这块内存。由于这块内存是在内核空间，而读取图像的数据是在应用程序中完成，因此需要将这一块内核空间的缓冲区配置以便应用程序可以访问。

Linux存在 3 种方式进行内核空间和用户空间的数据交互：Buffered I/O，Direct I/O和mmap。在Buffered I/O中，内核提供从用户空间拷贝数据和向用户空间拷贝数据的访问历程给驱动使用。在Direct I/O中，驱动可以通过kiobuf接口直接从用户空间缓冲区读写数据。第 3 种方法叫做mmap，驱动程序将一片内核空间的内存通过mmap内核调用映射到用户空间，应用程序可以直接对这块核心空间的内存进行读取[6]。由于应用程序需要直接读写一块用于DMA的核心空间内存，因此应该采用mmap。下面的代码将一块内核空间的内存映射到了用户空间：

```
//从这块内存的首地址开始映射
pos = (unsigned long)kernBuffer;
//得到首地址对应的页面
page = virt_to_phys((void *)pos);
//使用 remap_page_range 将这块内存映射到用户空间
remap_page_range(vma, start, page, size, PAGE_SHARED));
```

上面就完成了将一块内核空间地址映射到用户空间，在应用程序中还需要使用 mmap 系统调用得到这个用户空间的地址，如下：

```
userBuf = mmap (NULL, MMT_BUF_SIZE, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
```

之后应用程序就可以直接操作这块内存了，通过内存映射提高了数据传输的效率。

2.2 图像发送和接收

DMA 表示直接内存存取，指数据传输不需要 CPU 的干预自动完成。需要为 DMA 提供一块物理上连续的内存，传输时给出这块内存的首地址、传输数据的大小和传输卡接收数据的首地址。为 DMA 传输开设一个固定大小的缓冲区，如果图像大小小于此缓冲区，则一次 DMA 操作就可以完成一帧图像的传输；如果图像大小超过了缓冲区的大小，那么就需要将图像分块，进行多次 DMA 传输直到传输完毕。DMA 缓冲区的大小可以由下面的实验来决定：本实验分别对 80MB、100MB 和 120MB 3 个图像文件采用多个不同的缓冲区大小所得到的传输时间来加以比较。实验截取的时间是从 DMA 启动开始到延迟过程调用唤醒应用程序结束。对每个缓冲区大小的时间测定都作了 50 次重复性试验，最后得到平均值以消除传输时间的浮动影响，得到结果如图 3 所示。

横坐标表示缓冲区大小，单位是 MBytes，纵坐标为相应的时间，单位是 ms。可以看出，缓冲区越大，需要的时间越短，如果要最快的速度，那么只需要将缓冲区设置为和图像数据一样大即可。不过这是不可取的，因为内存资源通常是有限的，并且为 DMA 分配的内存要求是物理上连续的。这两个原因决定了不可能为 DMA 分配过大的内存，尤其当系统运行一段时间后，内存碎片的产生会导致分配一片连续的物理内存更加困难。

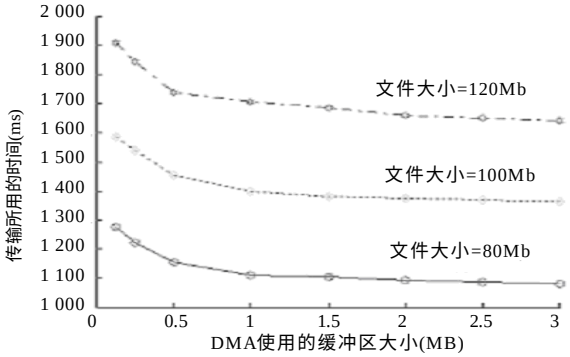

图 3 DMA 缓冲区大小对传输文件时间的影响分析

由图 3 可以看出，当缓冲区大小大于 2MB 之后，再增大缓冲区的大小时间的减少量已经不多，因此本系统采用 2MB 的缓冲区大小，既达到实时的要求，也没有过多使用系统资源。传输卡的最高带宽为 579.348Mbps。下面的代码分配了物理上连续的 2MB 的空间：

```
#define MMT_BUF_SIZE 0x200000
void *kernBuffer = __get_free_pages(GFP_DMA,
get_order(MMT_BUF_SIZE));
```

__get_free_pages 返回分配的这块内存在内核空间的首地址。下面的代码给出了如何将内核空间的地址转成可用于 DMA 传输的物理地址。

```
dmaSendAddr = dma_map_single(&pdev->dev, kernBuffer,
MMT_BUF_SIZE, DMA_TO_DEVICE);
```

然后通过设置 PCI 相应的寄存器就可以开始 DMA 传输这块内存的数据了，接收数据的方法类似。

当 DMA 传输结束后，传输卡会发出中断。如果注册了中段服务例程，操作系统就会调用它做些后续的处理。下面介绍在 Linux 下中断服务例程及延迟过程调用。

2.3 Linux 下的中断服务例程(ISR)和延迟过程调用(DPC)

当一次 DMA 传输完成之后，硬件中断产生，Linux 调用相应的中断服务例程，中断服务例程清除硬件中断信号位、unmap 内核空间地址到 DMA 硬件地址的映射并且插入一个延迟过程调用请求。延迟过程调用历程唤醒在等待 DMA 完成的信号，应用程序使用 poll 得到这个信号后就可以继续传输下一帧数据了。如此不断地给传输卡提供数据直到所有数据传输完毕。图 4 是中断的服务流程。

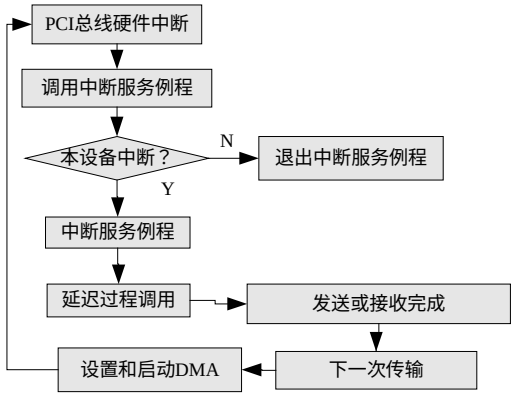

图 4 DMA 中断服务流程

首先得到设备的中断号：

```
(void) pci_read_config_byte(pdev, PCI_INTERRUPT_LINE, &pdev->irq);
```

这是通过读取 PCI 设备的配置空间完成的，这个函数将设备的中断号保存在了 pdev 结构的 irq 字段里，其中 pdev 是一个用于描述 PCI 设备的结构，原型为 struct pci_dev，在<linux/pci.h>中定义。

应用程序打开设备时需要注册中断服务例程：

```
request_irq(pdev->irq,pci9054_Isr,SA_SHIRQ,"pci9054",pci9054_ids);
```

当 PCI 传输卡发出中断之后，pci9054_Isr 就会被操作系统调用。下面是中断服务例程的主体：

```
irqreturn_t pci9054_Isr(int irqn, void *dev_id, struct pt_regs *regs)
{ // unmap 内核空间到 dma 空间的地址转换
 dma_unmap_single(&pdev->dev, dmaSendAddr, ulSendSize, DMA_TO_DEVICE);
 // 请求 dpc...在 Linux 下称作 tasklet，pci9054_tasklet 就是延//迟过程调用的名称
 tasklet_schedule(&pci9054_tasklet);
 // 返回 IRQ_HANDLED 表示中断已经被处理
 return IRQ_HANDLED; }
```

由于中断服务例程运行在很高的优先级上，当它运行的时候会把所有同优先级和比他优先级低的中断全部屏蔽，因此中断服务历程要尽快返回。这就要求将一些不是马上要处理的事情放到另外的地方处理，因此就有了延迟过程调用，意思是操作系统在合适的时间调用。在我们的系统中延迟过程调用将等待队列中的线程唤醒：

```
// 首先声明延迟过程调用例程
DECLARE_TASKLET(pci9054_tasklet, pci9054_do_tasklet, 0);
// 声明等待队列
static DECLARE_WAIT_QUEUE_HEAD(pci9054_wq);
// 下面是 dpc 例程的主体
void pci9054_do_tasklet(unsigned long unused)
{  // …
wake_up_interruptible(&pci9054_wq);
 // … }
```

这样当应用程序使用 poll 询问设备的状态时，当 dpc 例程被调用后，应用程序就不再被 poll 阻塞，如下所示：

```
struct pollfd pfd;
pfd.fd = fd;    // fd 是该 PCI 传输卡设备的描述符
pfd.events = POLLIN | POLLRDNORM;
retvalue = poll (&pfd, 1, -1);      // 当中断未发生时，应用程序//会被阻塞
if (pfd.revents == pfd.events)
// 中断发生，dpc 例程唤醒了等待队列里的线程
```

至此就完成了中断服务的功能，应用程序可以继续传输数据或者结束。

3 Windows 和 Linux 传输效率比较

为了将Linux的结果和Windows进行比较，我们使用Driver Studio[7]为传输卡也开发了用于Windows的驱动程序。在同一台机器上实验得出两个系统下传输时间的差异。实验采用的Linux内核为 2.6.9-11.EL，Windows的为xp-sp2。实验截取的时间为从将图像数据文件从硬盘读入为DMA开辟的缓冲区开始到延迟过程调用唤醒应用程序结束。表 1 示出了在传输同一文件(80MB)使用不同缓冲区大小时Windows和Linux所用时间的比较。

表 1 Linux 和 Windows 传输时间差异

缓冲区	Linux 用时	Windows 用时	时间差异
128kB	1 386.80ms	1 367.98ms	18.82ms
256kB	1 351.90ms	1 353.37ms	-1.47ms
512kB	1 302.70ms	1 331.44ms	-28.74ms
1MB	1 278.40ms	1 306.38ms	-27.98ms
2MB	1 269.90ms	1 298.90ms	-29ms

可以看到，Linux 只在缓冲区大小为 128kB 时所用时间多于 Windows，其他几种情况传输时间均少于 Windows。因此将 Linux 作为传输卡的主机操作系统并不会使其传输性能下降，在很多情况下反而比 Windows 有所提升。

4 结束语

本文研究并实现了一种高速实时的图像传输卡在 Linux 操作系统下的驱动程序，传输卡可以进行实时的图像传输，并根据实验确定了实现中的一些影响传输性能的关键参数的选取。对传输卡的传输性能在不同的系统下作了比较，根据实验得出Linux的传输性能在多数情况下要稍高于Windows。

参考文献

1 Shanley T, Anderson D. PCI System Architecture(4th Edition)[M]. Addison Wesley, 1999.

2 The Rampantly Unofficial Linus Travolds FAQ[Z]. http://www.tuxedo.org/~esr/faqs/linus/index.html, 2002.

3 Deitel H M. Operating Systems(2nd Edition)[M]. Addison Wesley, 1990.

4 Beck, Bohme, Dziadzka, et al. Linux Kernel Internals[M]. Addison Wesley, 1998.

5 Tsegaye M. A Comparison of the Linux and Windows Device Driver Architectures[J]. Operating Systems Review, 2004, 38(2): 8-33.

6 Corbet J, Kroah-Hartman G, Rubini A. Linux Device Drivers(3rd Edition)[M]. Addison Wesley, 2005.

7 Compuware. Numega DriverStudio[Z]. http://www.compuware.com. 2005.

（上接第 267 页）

如本系统中的答疑系统覆盖的范围不足，以及试题库的智能化不够高等都待我们去进一步完善和提高。

参考文献

1 黎炯宏. 工程制图多媒体教学系统的研究与开发[D]. 天津：天津科技大学, 2003.

2 方利伟. 基于 ASP 的 Web 考试系统[J]. 甘肃教育学院学报(自然科学版), 2002, 16(3).

3 Jones A R. 陈建春, 白　雁, 杨永平译. ASP.NET 与 C#从入门到提高[M]. 北京: 电子工业出版社, 2003: 310-405.

4 陈萱华. 基于ASP.NET的试题库系统[J]. 计算机应用, 2003, 23(1).