用户态驱动框架的研究与实现①

刘军卫[1]，李 曦[1,2]，陈香兰[1,2]，徐 军[1]

[1](中国科学技术大学 计算机科学与技术学院，合肥 230026)
[2](中国科学技术大学 苏州研究院，苏州 215123)

摘 要：在深入研究了 Linux 操作系统驱动模型的基础上，设计和实现了一种全新的用户态驱动框架 U2MDF（Unified User-Mode Driver Framework）。U2MDF 的核心思想是将传统的设备驱动分成内核态驱动模块和用户态驱动模块两部分，内核态驱动模块包含与性能密切相关的热点代码，如中断处理函数等；用户态驱动部分包含与性能无关的冷点代码，如设备的初始化等。以 RTL8139 网络设备为例，实现了 U2MDF 的原型系统，实验结果证明，U2MDF 在满足实际应用对性能要求的前提下，有效地减少了运行在内核态的驱动代码，基本上实现了驱动和内核的隔离，最终达到了提高操作系统整体可靠性的目的。

关键词：驱动模型；U2MDF；热点代码；可靠性

Research and Realization of a User-Mode Driver Framework

LIU Jun-Wei[1], LI Xi[1,2], CHEN Xiang-Lan[1,2], XU Jun[1]

[1](University of Technology and Science of China, Anhui University, Hefei 230026, China)
[2](Suzhou Institute for Advanced Study, University of Technology and Science of China, Suzhou 215123, China)

Abstract: Based on the research of Linux Device Driver Model, this paper proposes and implements a new user-mode device driver framework called U2MDF(Unified User-Mode Driver Framework). The core idea of U2MDF is splitting the traditional device driver into two parts. One called kernel-mode driver component contains the hot code related to performance or critical paths, such as interrupt handler, and the other called user-mode driver component contains the performance-independent code, such as the device initialization. To take network device RTL8139 for an example, the prototype system is implemented. Experiments show that U2MDF reduces the driver code that runs in the kernel mode effectively and isolates the driver and kernel roughly. Therefore it ultimately improves the reliability of operating systems.

Key words: driver model; U2MDF; hot code; reliability

随着信息化技术的发展，Linux 已经被广泛地应用到服务器和桌面系统，近年来，随着嵌入式系统应用的持续升温，Linux 也开始应用于嵌入式领域，逐步成为通信，工业控制，航天军工，医疗卫生、消费电子等领域的主流操作系统。研究表明，70%的 Linux 内核代码来自于设备驱动[1]，现在主流的桌面操作系统大约包含 35,000 个设备驱动程序[2]。

性能和可靠性是衡量 Linux 操作系统的两个关键参数。随着硬件技术的发展，相对于性能来说，可靠性被公认为当今操作系统研究的热点之一[2-5,13]，尤其在服务器和嵌入式领域，这一需求表现的更为迫切。斯坦福大学的一项研究发现，在 Linux 操作系统中，由设备驱动导致的系统崩溃率是操作系统其他模块的 3 到 7 倍[1,8,10]。

① 收稿时间:2011-03-25;收到修改稿时间:2011-04-23

在传统的 Linux 驱动模型中，为了兼顾系统性能和 I/O 指令的特权操作要求，设备驱动往往与操作系统内核运行在同一个地址空间，具有内核的所有权限（例如可以直接访问所有物理内存、任意访问和操作内核其他模块等）。一般而言，操作系统对运行在内核态的代码是无条件信任的，因此，设备驱动内一个小小的 bug，也将会导致一个系统范围内的故障。另外，内核驱动编程要遵循严格的限制，驱动开发、调试以及测试的工具匮乏，很难保证驱动的可靠性。

U^{2}MDF 是一种全新的用户态驱动框架[8-12,14]，即将部分驱动程序运行在用户模式，使其仅能访问用户地址空间，但可以像内核驱动（Kernel-Mode Driver Framework）那样具有直接访问硬件设备和执行 I/O 特权指令的权利。由于采用了用户态驱动的设计思想，因此，U2MDF 具有以下优势：

① 实现了驱动和内核的隔离，有效地减少了运行在内核态的驱动代码，显著地提高了操作系统的可靠性；

② 用户空间的驱动开发，可以采用许多优秀的开发工具和函数库，如 gdb，Interl VTue，glibc 库等；

③ 良好的性能，完全可以满足实际应用的要求；

④ 统一的编程框架，可适用于 Linux 操作系统下绝大多数驱动；

1　U^{2}MDF体系结构

U^{2}MDF 将传统的设备驱动划分为用户态驱动模块(User-Driver，简称 UD)和内核态驱动模块(Kernel-Driver，简称 KD)。KD 主要包含性能相关的热点代码和关键路径（中断处理函数，软中断，频繁调用的函数，数据拷贝的函数），以可加载内核模块（LKM）的方式运行在内核地址空间。 UD 主要包含性能无关的冷点代码（设备的初始化、配置、控制以及出错处理以及偶尔被调用的函数等），以进程或可加载动态共享库的形式运行在用户地址空间。KD 又分为内核驱动核心模块（Kernel-Driver CORE，简称 KD-CORE）和内核驱动通信模块（Kernel-Driver COM，简称 KD-COM），KD-CORE 完成内核驱动模块的核心功能，如中断处理，为 UD 提供服务等；KD-COM 主要用于响应和管理 KD 与 UD 之间的通信。与 KD 类似，UD 也分为用户驱动核心模块（User-Driver CORE，简称 UD-CORE）和用户驱动通信模块（User-Driver COM，简称 UD-COM），UD-CORE 完成用户驱动模块的核心功能，如设备初始化，设备配置，出错处理等功能；UD-COM 主要负责与 KD 之间的交互，如请求 KD 提供服务，执行来自 KD 的函数调用请求等。U^{2}MDF 体系结构如图 1 所示。

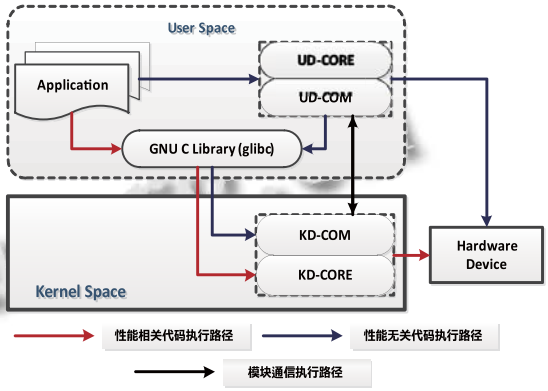

图 1　U^{2}MDF 体系结构图

U^{2}MDF 体系结构图中的红色箭头路径是性能相关的热点代码的执行路径，绝大部分是与数据发送和接收有关的操作，因此也可以称为数据路径（Data Path），仍然保留在内核空间执行，最大限度地保证了 U2MDF 驱动的性能；性能无关的冷点代码的执行路径为 U^{2}MDF 体系结构图中的蓝色箭头路径，绝大部分是与设备控制操作有关，因此也可以称为控制路径（Control Path），被隔离到用户地址空间运行，实现了驱动和内核的隔离，提高了整个操作系统可靠性。U^{2}MDF 的设备驱动的内核接口并没有变化，因此，兼容市场上流行的绝大多数 Linux 操作系统。另外，对于传统的设备驱动只需要增加 UD 和 KD 之间通信的代码就可以转变成符合 U^{2}MDF 的用户态驱动；对于新开发的驱动，可以直接采用 U^{2}MDF 驱动设计思想，大大降低了驱动开发的复杂度。

2　U^{2}MDF的设计与实现

为了展示 U^{2}MDF 驱动框架的可行性、性能和可靠性，本文基于 Intel X86 平台，Linux-2.6.35 内核，以 RTL8139 网络设备为例，实现了 U^{2}MDF 驱动框架的原型系统。按照 U^{2}MDF 驱动框架的设计思想，基本确定以下实现方案：

① 内核驱动核心模块 KD-CORE 以可加载内核模

块的形式来实现；

② 内核驱动通信模块 KD-COM 采用优化的 Netlink 机制来实现;

③ 用户驱动核心模块UD-CORE以用户进程的形式来实现;

④ 用户驱动通信模块 UD-COM 采用优化的 ioctl 机制来实现;

2.1 U²MDF 网络设备的 I/O 访问

要实现在用户态直接访问 RTL8139 的寄存器或 I/O 端口，按照 U²MDF 的设计思想只要完成以下两个目标即可：

① 使用户态驱动进程获得访问设备对应的I/O端口或寄存器的权限；

② 使用户态进程的获得硬件设备对应的I/O基址或寄存器基址；

通过 iopl()和 ioperm()两个系统调用可以实现目标○1，在 U²MDF 的原型系统中采用了 iopl(3)系统调用，其功能是允许调用进程获得所有的 65535 个端口的权限，其函数原型如下：

int iopl (int level); //level 为端口权限的级别

通过读取/proc/ioports 文件可以实现目标○2，即获得设备对应的 I/O 端口或寄存器基址。

2.2 U²MDF 网络设备的通信机制

按照 U²MDF 的设计思想，KD 和 UD 必须协同工作才能完成传统驱动的功能，那么，U²MDF 的实现中首先需要考虑的就是如何实现 KD 和 UD 之间的通信，从实现的角度来讲，即 Linux 系统中内核态与用户态之间的通信。通信本质上是不同上下文的切换，众所周知，上下文的切换必然会导致巨大的性能损失，因此，选择何种交互方案是能否保证 U²MDF 性能的关键。KD 和 UD 之间的通信流程如图 2 所示。

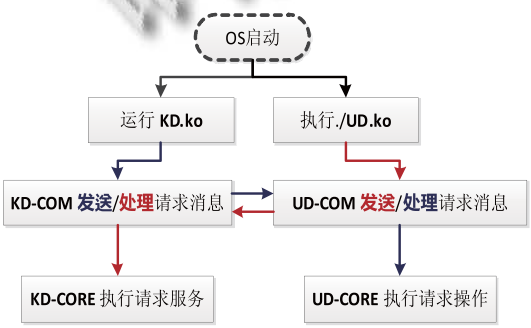

图 2 U²MDF 通信流程图

2.2.1 U²MDF 网络设备的数据通信机制

U²MDF 网络驱动采用 ioctl()机制实现了 KD 和 UD 之间的通信，对于 UD 向 KD 模块或 Linux 内核发起请求或 UD 与 KD 之间需要阻塞的服务，最终交由交互管理模块中注册的 ioctl()的实现函数来实现，目前，U²MDF 网络驱动支持的 UD 向 KD 发起的请求可以分为：

① 计算 UD 模块中虚拟地址对应的物理地址的请求 VIRT2PHY；

② 向 Linux 内核空间提交当前用户态驱动进程信息的请求 UP_INFO;

③ 获取 KD 模块中 DMA 缓冲区的物理地址的请求 DMA_PHY；

④ UD 模块请求 KD 驱动模块某项服务的功能，如 OPEN，CLOSE 等；

2.2.2 U²MDF 网络设备的中断通知机制

在按照 U2MDF 的设计思想实现的驱动中，KD 和 UD 之间最常见的一种通知机制就是如何将硬件产生的中断通知给用户空间的程序，这在传统的驱动程序中是不可能实现的，因为中断的响应和处理过程都是在内核空间实现的，用户空间的程序不可能也没有办法响应和处理硬件产生的中断，但从理论上讲，U²MDF 目前至少有两种方法可以将硬件中断通知给用户程序。

第一种方法是将中断机制映射到 Linux 信号机制上，即一旦设备就绪，则主动通知应用程序，应用程序不需要查询设备的状态，这非常类似于硬件上中断的概念，这种机制就是“信号驱动（SIGIO）的异步 I/O”[15,16]。为了在用户空间能处理一个设备释放的信号，必须完成以下三项工作：

① I/O 命令 F_SETOWN 设置文件拥有者为当前进程；

② I/O 命令 F_SETFL 设置设备文件支持 FASYNC；

③ 通过 signal()函数连接信号和信号处理函数；

为了使设备支持异步通知机制，驱动程序也需要完成以下三项操作：

⑤ 支持 F_SETOWN 命令，在控制命令中设置 filp->f_owner 为对应进程的 ID；

⑥ 支持 F_SETFL 命令，每当 FASYNC 标志改变时，驱动程序的 fasync()函数得以执行；

⑦ 在硬件设备产生时，用 kill_fasync()函数激发响应的信号。

第二种方法是将中断映射到文件描述符。在 Linux 操作系统的/proc 文件系统下，每一个中断都对应一个目录，该目录下包括一些属性文件，如 cpu 亲和性设置文件等。按照 U²MDF 思想设计的驱动通过在中断号对应的目录下增加一个新的属性文件，用来标识是否产生了该中断号对应的中断。当用户空间程序对该文件执行 read()操作时，内核在对应的信号量上执行 down()操作，目的是阻塞该 read()操作直到中断的产生；当硬件产生中断时，首先在内核中断处理函数中屏蔽该中断，然后增加中断映射的文件描述符中的中断计数，最后在对应的信号量上执行 up()操作，使之前阻塞的 read()操作返回，返回值为文件描述符中的中断计数，如果返回值大于 0，说明产生了硬件中断，则转而去执行用户空间定义的中断处理函数。需要注意的是，将中断映射到文件描述符的设计思想仅仅对电平触发的中断是有效的，不过幸运的是，所有的 PCI 中断都是电平触发的。

在 U²MDF 的原型系统中，实现了将中断映射到 Linux 信号和将中断映射到文件描述符两种中断通知机制，实验结果证明将中断映射到文件描述符的性能优于将中断映射到 Linux 信号的机制。

2.2.4 U²MDF 网络设备的 DMA 缓冲区

U²MDF 网络驱动程序分配内存页，并映射这些内存页到指定用户进程的地址空间[15,16]，就可以实现 Linux 内核和用户态驱动共享内存，具体的实现方法参见如下代码：

```
/*驱动程序中关键代码*/
kaddr = kmalloc(size);
phy_addr = addr – PAGE_OFFSET;
/*应用程序中关键代码*/
fd = open(“/dev/mem”, O_RDWR);
uaddr = mmap(phy_addr, length, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0 );
```

3 测试结果及性能评价

网络吞吐量直接反映了网络驱动的性能，是网络设备驱动开发者最为关心的指标，另外，CPU 占用率直接反映了驱动对操作系统带来的开销，如不必要的上下文切换等都会造成 CPU 的占用率升高，因此，它也是衡量网络驱动的重要指标之一。本设计采用 netperf 网络性能测试工具对 Linux 内核中原有的 8139too 驱动和按照 U²MDF 设计思想改写的 8139too-U²MDF 驱动的吞吐量和CPU 占用率进行了测试。测试环境的 client 和 server 端都采用 KVM 虚拟的 Linux 操作系统，具体配置如表 1 所示。

表 1　client 和 server 端配置参数

配置	参数
Model Name	Qemu Virtual CPU version 0.12.5
CPU MHZ	2527.224
Cache Size	4094KB
RAM Size	512MB
Network Device	RTL8139
Linux Kernel	2.6.35-x86_64

测试用例基于 TCP 协议，在 client 和 server 端之间连续的发送 16384 字节的测试分组，测试 client 和 server 之间的网络吞吐量，client 端的 cpu 占用率和 server 端的 cpu 占用率，对应的测试结果分别如图 3，图 4，图 5 所示。测试结果表明，按照按照 U²MDF 设计思想改写后的网络驱动的吞吐量基本与原驱动相当，cpu 占用率仅有少许升高，在实际应用中基本可以忽略不计。

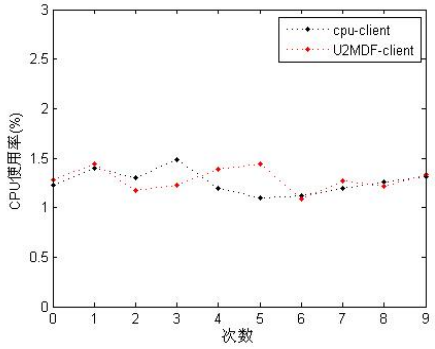

图 3　Client 端 CPU 占用率

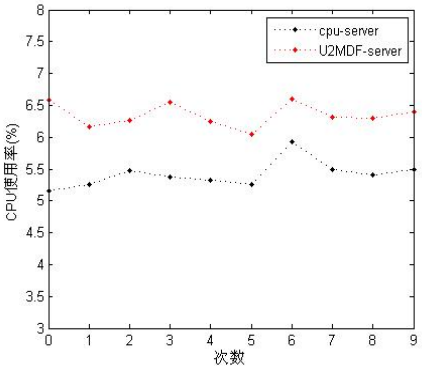

图 4　Server 端 CPU 占用率

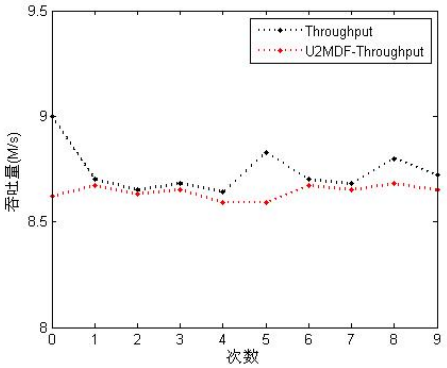

图 5 网络吞吐量

U²MDF 的主要目标就是通过减少运行在内核态的驱动代码以达到提高 Linux 操作系统可靠性的目的。在分析了 Linux 内核中原有 8139too 网络驱动和按照 U²MDF 设计思想改写的 8139too-U²MDF 驱动的 SLOC（Source Lines Of Code）之后，给出统计数据如表 2 所示。表 2 只统计了驱动中的函数代码，并没有包括全局变量定义，宏，头文件等代码。统计结果表明按照 U²MDF 改写的 8139too 网络驱动可以将多达 65.4% 的代码放到用户态，从根本上隔离了驱动代码可能产生的故障，能够显著的提高 Linux 操作系统的稳定性。

表 2 网络驱动代码统计数据

	内核态 SLOC	用户态 SLOC
8139too	1615	0
8139too-U²MDF	559	1056

4 结论和展望

本文提出了一种新的用户态驱动框架 U²MDF，详细的阐述了 U²MDF 的实验原理和实现细节，并以网络设备 RTL8139 设备为例实现了 U²MDF 的原型系统。实验结果证明，U²MDF 在满足实际应用对性能要求的前提下，有效的减少了运行在内核态的驱动代码，基本上实现了驱动和内核的隔离，最终达到了提高操作系统整体可靠性的目的。同时，U²MDF 为驱动开发者提供了一种用户态驱动开发的统一框架，并且适用于 Linux 操作系统中绝大部分的设备驱动。

Linux 操作系统中设备驱动种类繁多，数量巨大，因此，下一步工作的和研究的重点是将 Linux 操作系统中其他典型的设备驱动改写为符合 U²MDF 的用户态驱动。

参考文献

1 Chou A, Yang J, Chelf B, Hallem S, Engler D. An empirical study of operating systems errors. Proc. of the Eighteenth ACM Symposium on Operating Systems Principles. ACM, 2001, 73–88.

2 Swift MM, Bershad BN, Levy HM. Improving the reliability of commodity operating systems. ACM Trans. on Computer Systems (TOCS), 2005,23(1):77–110.

3 LeVasseur J, Uhlig V, Stoess J, Stefan Götz. Unmodifed device driver reuse and improved system dependability via virtual machines. Proc. of the 6th Conference on Symposiumon Opearting Systems Design & Implementation, Berkeley, CA, USA.USENIX Association. 2004, 6: 2–2.

4 Swift MM, Martin S, Levy HM, Eggers SJ. Nooks: An architecture for reliable device drivers. Proc. of the 10th Workshop on ACM SIGOPS European Workshop. ACM, 2002. 102–107.

5 Witchel E, Rhee J, Asanovi' c K. Mondrix: Memory isolation for Linux using Mondriaan memory protection. Proc. of the Twentieth ACM Symposium on Operating Systems Principles. ACM, 2005: 31–44.

6 Ball T, Bounimova E, Cook B, Levin V, Lichtenberg J, McGarvey C, Ondrusek B, Rajamani SK, Ustuner A. Thorough static analysis of device drivers. ACM SIGOPS Operating Systems Review, 2006, 40(4):73–85.

7 Padioleau Y, Lawall JL, Muller G. Understanding collateral evolution in Linux device drivers. Proc. of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, ACM, 2006: 59–71.

8 Ganapathy V, Renzelmann MJ, Balakrishnan A, Swift MM, Jha S. The design and implementation of microdrivers. Proc. of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM, 2008: 168–178.

9 Chubb P. Get more device drivers out of the kernel. Ottawa Linux Symp, Citeseer, 2004: 149–161.

10 Ganapathy V, Balakrishnan A, Swift MM, Jha S. Micro-dvers: A new architecture for device drivers. Proc. of the 11th USENIX Workshop on Hot Topics in Operating Systems, USENIX Association, 2007. 1–6.

(下转第 90 页)

4 结语

本文在首先将自适应控制和鲁棒控制算法融合到传统迭代控制方法中去，以提高机器人在强干扰及不确定情况下的控制精度，由图 3 和图 4 可知算法能够跟踪给定的运动轨迹,但是在任务发生变化时,由图 5 可知，收敛所需迭代次数较大，迭代到 12 次时才收敛，为进一步改善系统的实时性，再采用快速简单的 FCMAC 加以前馈改善初始值，当任务轨迹改变时加入 FCMAC 仅用 4 次就可收敛，在更短的时间内达到了精度的要求，在时间效率上较之前提高了 75%。

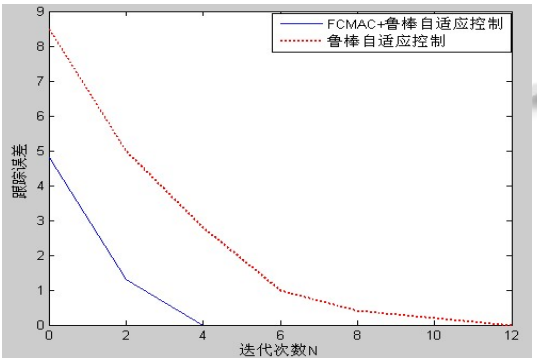

图 5 2 种算法效果对照图

参考文献

1 谢胜利,田森平,谢振东,等.迭代学习控制的理论与应用.北京:科学出版社.

2 Arimoto S, Kawamura S, Miyazaki F. Bettering Operation of robotics by learning. Journal of Robotic System, 1984,1(2): 123–140.

3 徐敏,林辉,刘震.可变学习增益的迭代学习控制.控制理论与应用,2007,24(5):857–861

4 姚仲舒,王宏飞,杨成梧.一种机器人轨迹跟踪的迭代学习控制方法.兵工学报,2004,25(3):330–334.

5 Zhao YJ, Dong XJ. A new solution to the inverse position analysis of the redundant serial robot. Journal of Shanghai Jiaotong University (Science), 2010, 15(5): 610–614.

6 侯海军,雷勇,叶小勇.基于模糊 CMAC 网络的非线性自适应逆控制.系统仿真学报,2008,20(8):2039–2042.

7 孙炜,王耀南.模糊 CMAC 及其在机器人轨迹跟踪控制中的应用.控制理论与应用,2006,2(1):38–42.

8 程启明,王勇浩.模糊 CMAC 神经网络控制系统及混合学习算法.电机与控制学报,2006,10(2):216–221.

9 Morris J, Zhang J. Performance Monitoring and Batch to Batch Control of Biotechnological Processes. 2009, 218: 281–310.

10 Levskii MV. Optimal control of reorientation of a spacecraft using free trajectory method,2011,49(2):131–149.

11 Rose M. Automated Generation of Efficient Real-Time Code for Inverse Dynamic Parallel Robot Models. Springer Tracts in Advanced Robotics, 2011,67:39–57.

(上接第 71 页)

11 Dike J. A user-mode port of the Linux kernel. Proc. of the 2000 Linux Showcase and Conference, 2000, 2.

12 Elson J. Girod L. FUSD-a linux framework for user-space devices. University of California, Los Angeles, Unpublished technical report, 2003.

13 Shen YT, Elphinstone K, Heiser G. User-Level Device Drivers: Achieved Performance.

14 Zhou F, Condit J, Anderson Z, Bagrak I, Ennals R, Harren M, Necula G, Brewer E. SafeDrive: Safe and recoverable extensions using language-based techniques. Proc. of the 7th Symposium on Operating Systems Design and Implementation. USENIX Association, 2006. 45–60.

15 Bovet D, Cesati M, Oram A. Understanding the linux kernel. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2002.

16 Love R. Safari Tech Books Online. Linux kernel development, Novell Press USA, 2005, 50.