单一内核操作系统设备驱动程序缺陷研究

秦 莹 戴华东 颜跃进

(国防科学技术大学计算机学院 长沙 410073)

摘 要 设备驱动程序是操作系统内核中代码量最大、缺陷最多的组件。单一内核操作系统中驱动程序处于内核态执行,驱动程序缺陷引发的错误往往直接导致内核崩溃,极大地降低系统可用性。分析了单一内核驱动程序存在的缺陷并将其按照位置分类,介绍了驱动程序缺陷解决方法。

关键词 单一内核,操作系统,设备驱动,缺陷

中图法分类号 TP314 **文献标识码** A

Study on Defects Relate to Device Driver in Monolithic Kernel of Operating System

QIN Ying DAI Hua-dong YAN Yue-jin

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract Device drivers are major and easier faulty part of kernel. Especially in monolithic kernel of operating system, device drivers execute in supervisor mode, defects in device drivers usually result in kernel panic, greatly degrade the availability of operating system. Defects in device drivers were analyzed and sorted, methods of reducing defects in device driver were introduced.

Keywords Monolithic kernel, Operating system, Device driver, Defect

1 引言

随着社会信息化程度的提高,运行于各类计算机及电子设备中的操作系统逐渐成为与日常生活息息相关的组成部分。操作系统可靠性成为与性能同等重要的追求目标。

主流操作系统如 Unix, Linux 采用了单一内核结构,设备驱动程序等扩展功能模块和内核核心代码同处于核态运行。这种单一内核结构在带来性能优势的同时也带来了可靠性隐患:设备驱动程序以任务队列处理线程和中断线程的形式处理用户请求和设备应答。中断处理和任务队列处理这两类内核线程一旦因程序缺陷(defect)而改变正常运行轨迹,将会导致操作系统内核运行混乱,进而引发内核崩溃。

斯坦福大学研究表明[1],Linux 驱动程序代码缺陷的出现频度(以平均每万行代码存在的缺陷数计)是 Linux 内核其他部分代码的 3～7 倍;而据美国 Coverity 公司报告[2],2009 年每万行 Linux 内核代码存在 1.27 个缺陷,据此可推算在 Linux 操作系统中每万行驱动程序存在 4～8 个缺陷。Windows Server 缺陷统计报告显示[3],Windows Server 设备驱动程序缺陷造成 87%的系统崩溃;即使是微软最新推出的 Windows7 操作系统,驱动程序仍然是制约操作系统可靠运行的主要障碍。

本文对单一内核操作系统中设备驱动程序缺陷进行分析并根据缺陷在驱动程序中所处的位置进行分类;在该分类基础上,分析国际上针对不同类型缺陷提出的解决方法,提出存在的问题和进一步研究方向。

2 驱动程序缺陷分析和分类

2.1 单一内核操作系统的内核和驱动结构

目前单一内核操作系统的内核采用模块化设计:内核核心开发者实现处理器调度、内存管理、并发控制机制(锁、信号量)等核心功能模块;采用动态加载内核模块机制,集成由第三方开发的设备驱动、文件系统等扩展功能模块。

设备驱动程序是重要内核扩展,表现在两个方面:①代码量大。管理 I/O 硬件资源是内核的主要任务,为驱动种类繁多的 I/O 硬件,内核集成大量驱动代码;驱动程序代码在 Linux2.6 内核中占到了 50%以上。②关系复杂。驱动程序和设备之间通过硬件协议完成数据交互;驱动程序还要和操作系统内核其他模块交互,如使用内核并发控制机制、内核内存分配功能及为文件系统提供数据服务等;驱动程序以内核线程形态存在,同其他内核线程竞争计算、存储等资源。

图 1 给出内核组成示意图,说明驱动程序在单一内核操作系统中的位置。虚线间的部分为操作系统内核,驱动位于内核子系统和设备之间,实现内核核心、设备之间交互代理功能。

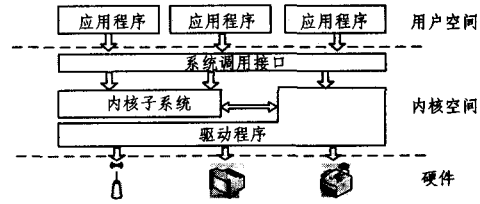

图 1 内核组成示意图

到稿日期:2010-05-07 返修日期:2010-08-29 本文受国家 863 计划重大项目(2008AA01A203),核高基重大专项(2009ZX01040-001-001)资助。

秦 莹(1974－),女,博士生,副研究员,主要研究方向为系统软件,E-mail:qy123@nudt.edu.cn;戴华东(1975－),男,博士,副研究员,主要研究方向为系统软件、容错;颜跃进(1976－),男,博士,助理研究员,主要研究方向为系统软件、容错。

2.2 驱动程序缺陷分类

作为软硬件接口的驱动程序，其缺陷涉及范围广、影响力强。因误用操作系统内存管理、线程管理接口，引发内存泄漏；因数组越界等程序语言缺陷，引发段错误；因未能正确处理硬件瞬时故障，引发运行挂起，导致系统失去响应。

驱动程序从逻辑上可划分为 3 部分：与内核子系统交互部分、与硬件系统交互部分、进行内部处理部分。Linux 内核 2002－2008 年驱动程序缺陷统计表明，驱动程序缺陷分布在驱动程序各逻辑组成部分。根据缺陷分布位置不同，将设备驱动程序缺陷划分为驱动内核接口、驱动程序内部、驱动设备接口 3 类缺陷。

2.2.1 驱动内核接口缺陷

驱动内核接口缺陷指位于驱动和内核接口处的缺陷，位置如图 2 所示。

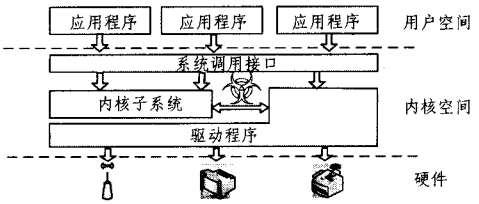

图 2 驱动内核接口缺陷

以内核模块方式加载并运行于内核空间的设备驱动，通过共享内存以及函数调用方式同内核核心模块交互，耦合度高。驱动程序与核心模块之间存在双向依赖关系：驱动程序需使用内核子系统的线程管理、内存管理和并发控制等功能函数；与设备 I/O 相关的系统调用需向驱动程序传递请求数据，并读取驱动程序返回数据。按照功能不同，驱动程序常用内核函数可分为以下几类：

①并发控制类。驱动程序以工作线程和中断线程方式运行于内核，和其他内核线程竞争使用处理器、内存、I/O 中断和设备端口等系统资源。为保证系统运行时逻辑正确，驱动程序使用自旋锁、信号量等并发机制保证资源互斥使用。

实现并发机制是内核高效运行的必然要求，近几年来多核处理器的普及进一步加深了这一要求的迫切性。为了提高并发执行效率，内核子系统通常实现多种并发机制，以应对不同上下文环境。Linux 操作系统就提供了 Spinlock，Rcu 等多种并发机制。

第三方驱动程序开发者对并发机制不够熟悉，误用并发机制，是内核缺陷的主要原因。主要误用有：驱动程序申请、释放锁的顺序不正确；使用未初始化锁；释放未申请锁。

②动态内存类。内存是计算机系统中最易短缺的资源，按需动态分配是操作系统内核以有限内存支持大量应用的有效方法。内核内存管理系统提供动态内存分配和释放函数在内核堆中获取可用内存。例如 Linux 内核提供 kmalloc 支持动态申请内存，提供 kfree 函数支持释放内存。

驱动程序中工作队列以动态增删链表形式存在，驱动程序在运行过程中动态申请内存，以存储运行过程中产生数据并被要求在用后释放。程序员疏忽会导致申请后不释放的情况出现，引发内存泄漏；因程序员疏忽导致多次调用 kfree 释放同一块内存的情况，引发指针错误。2009 年 Coverity 公司对 280 种开源项目的代码进行扫描发现，资源泄漏和释放空指针缺陷所占比例列程序缺陷排名前两位[2]。

③定时器类。驱动程序使用定时器来确定某项任务时间是否到期。由于程序员疏忽导致驱动程序对定时器函数的误用主要有：使用未初始化定时器、使用定时器数超过上限。

2.2.2 驱动程序内部缺陷

驱动程序内部缺陷是指驱动程序内部逻辑缺陷，位置如图 3 所示。

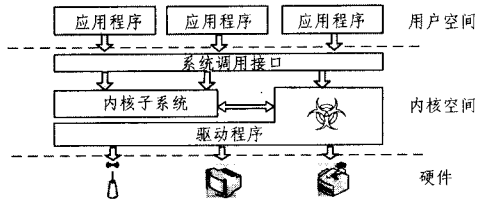

图 3 驱动程序内部缺陷

表 1 取自 Coverity 公司 2009 年对 280 种开源程序缺陷的统计报告[2]，缓冲区溢出、忽略表达式等常见程序缺陷占有相当比例。

表 1 缺陷类型和出现比率

缺陷类型	出现比率
释放空指针	27.81
资源泄露	23.34
忽略表达式	9.71
未初始化数据读	8.41
测试之前使用(空指针)	8.35
释放后使用	5.91
缓冲区溢出(静态分配)	5.79
不安全的使用返回的空指针	5.3
不安全的使用返回的负值	3.9
类型和分配大小不符	1.1
缓冲区溢出(动态分配)	0.21
测试之前使用(负值)	0.18

同普通应用程序一样，采用 C/C++语言实现的设备驱动代码，存在除零、数组越界、指针错误等程序缺陷。Coverity 公司对 Linux2.6 内核进行的统计表明[2]，缺陷为 1.27 个/万行。新南威尔士大学操作系统内核研究小组对 Linux 程序统计发现语言引发程序缺陷比例为 23%[6]。

除了驱动程序对队列、数组等数据结构操作不当引发程序缺陷，程序控制流缺陷也是重要缺陷来源。微软公司根据对 Windows 缺陷跟踪发现，驱动程序缺陷分布于中断请求完成操作、请求取消操作、请求处理操作等处。

2.2.3 驱动设备接口缺陷

驱动设备接口缺陷是指位于驱动程序和硬件设备接口中的缺陷，位置如图 4 所示。

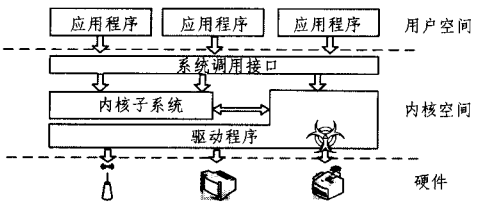

图 4 驱动设备接口缺陷

文献[15]指出进入后 CMOS 时代，随着芯片中晶体管密度持续增加，设备芯片老化和放射性引发单粒子翻转(single event updates，SEU)等软错误(soft error)、硬件 FIFO 溢出等硬件设计缺陷，导致硬件故障的情况日益突出。为实现设备功能扩展灵活性，目前大量设备中集成了固件，某些设备固件

缺陷也会表现为硬件故障。微软公司报告[3]显示，Windows服务器操作系统遇到磁盘或是网卡故障，通常由芯片瞬时故障导致。

与频繁出现的设备故障相对，设备驱动程序开发者往往忽略瞬时硬件故障和固件故障的存在。文献[5]对Linux设备驱动程序分析发现，由于设备驱动程序编写者在进行程序编写时，通常假定硬件总能正常工作，处理流程仅仅考虑对设备正常情况的应对，忽略对硬件瞬时故障或间歇性故障的处理，使得此类故障发生时驱动程序无法作出正确反应，导致系统崩溃。

实例1　下段代码取自Linux2.6.18内核网卡设备驱动3c59x.c[5]。如果由于硬件故障导致ioread16无法从设备中获得正确的返回值，程序将会陷入死循环，导致系统挂起。

```
while (ioread16 (ioaddr + Wn7 _ MasterStatus)) &
0x8000)
```

实例2　下列代码同样取自Linux2.6.18内核驱动。在例中，从硬件寄存器中取得的值将作为数组下标，由于下标值未经过验证，小于零或大于数组限制的下标均会导致数据引用错误。

```
static void __init attach_pas_card(…) {
if((pas_model=pas_read(0xFF88)))
{ char temp[100];
sprintf(temp,"%s rev %d",
pas_model_names[(int) pas_model],
pas_read(0x2789));
}
```

上述程序实例表明，驱动程序对硬件设备接口处理不完善，使硬件设备故障通过驱动程序扩散到内核，导致内核运行出现异常。由于程序看起来并无逻辑错误，硬件故障引发的驱动程序缺陷具有隐蔽性较高的特点。

2.3　小结

驱动程序缺陷对操作系统性能和可靠性均会产生影响，进行缺陷分类将有助于有的放矢地进行精确的故障处理。

3　驱动程序缺陷解决技术

对操作系统进行架构上改进[16]，将代码量巨大且容易出错的设备驱动实现在内核核心之外，是较好选择。上述改进已在一些系统中付诸实施：微软提出的UDMF[12]，采用用户态驱动来减少驱动对操作系统的影响；微软新一代操作系统Singularity[13]，通过采用类型安全语言减少驱动程序自身缺陷，并采用微内核架构避免驱动缺陷影响内核；Minix操作系统[14]以微内核进程方式运行驱动，并采用静态内存分配减少因动态内存分配对操作系统可靠性影响。但上述方法无法解决的主要问题是：目前大量部署的操作系统仍采用驱动程序运行在核态的单一内核形式；精确定位并消除驱动程序缺陷是单一内核操作系统驱动开发和部署过程中必须解决的问题。

基于上一节对驱动程序缺陷的分类，本节介绍目前国际上提出的主要解决方法。

3.1　基于静态程序检测分析的驱动缺陷定位技术

微软SLAM项目[8]始于2000年，2007年以项目成果为基础开发的SDV工具已经用于Windows驱动程序验证。SLAM项目主要基于反例制导谓词抽象精化(CEGAR)方法进行程序静态分析[9]。CEGAR方法包括抽象、模型检测、精化3个过程。抽象过程提取与待验证属性相关程序骨干，待验证属性由驱动程序缺陷决定；模型检测过程检查程序骨干是否存在与待验证属性相符合的路径情况(称为反例)，因程序骨干和实际程序相比存在信息损失，在模型检测过程发现反例时，还需验证反例是否存在于真实程序中。若反例存在于真实程序中，则发现程序缺陷；否则对程序骨干进行精化，加入可剔除虚假反例的控制逻辑，生成下一轮程序骨干，继续检测过程。CEGAR是逐步求精过程，可有效解决模型检测存在的空间爆炸问题，可用于验证大型程序。SLAM实现程序属性描述语言Slic，支持用户灵活配置待验证程序属性。Blast[10]在SLAM基础上进行优化并进一步扩展了可验证程序规模，将其用于Linux内核验证。DDverify系统[4]，采用SLAM类似思想，针对Linux驱动进行验证；DDverify系统实现Linux内核框架，为驱动程序提供执行场景，并将驱动程序属性验证代码内置于内核框架中；使用SATABS和Boppo作为程序抽象精化器和模型检测器。由于不支持类似Slic的属性描述语言，DDverify存在无法灵活扩展属性验证项的缺陷。

斯坦福大学[1]在编译器控制流、数据流分析机制中扩展程序属性分析功能，以对包括驱动程序在内的各类程序源代码进行属性检测。基于斯坦福大学研究成果的工具已被美国国土安全局用于检查开源代码，以提高质量[2]。

CEGAR解决空间爆炸问题的代价是损失程序精度；基于CEGAR的程序模型检测和基于编译的程序分析都需要有准确定义的程序属性，定义全部程序属性是不可判定问题。两方面因素交织，上述程序检测、分析的结果不完备也不可靠，减少误报漏报现象是努力方向。上述工作集中于发现驱动程序程序固有缺陷、内核接口调用缺陷，对设备硬件接口缺陷关注少。

3.2　动静结合的驱动设备接口缺陷处理技术

驱动程序特殊性在于包含与硬件交互代码。这种特殊性使得某些时序要求不仅仅性能相关而且正确性相关。若多次尝试可以解决一般瞬时故障，永久性故障则需加入尝试次数限制；编译优化的死代码删除环节不得删除I/O指令前后的空指令循环，因为循环是为了匹配处理器和设备处理速度特意加入的，缺少会导致程序错误。

Carburizer是威斯康辛大学实现的一种面向硬件故障的驱动程序加固设施[5]。它采用静态分析和动态处理结合方式：在程序编译过程中自动识别与硬件交互语句并自动加入判断和故障恢复调用语句；程序运行时按需调用内核集成的故障恢复模块，自动实现硬件故障处理。Carburizer在编译时进行敏感代码自动修正，程序修正效率优于以手工进行驱动程序加固[11]的方式；采用动态故障处理，提高了硬件故障容忍能力。

3.3　基于形式化方法的驱动自动生成技术

前两种方法均是以修补驱动程序缺陷为目标。针对这一情况，新南威尔士大学进行了利用形式化方法生成完备、可靠驱动程序的尝试。Dingo[6]是该大学设计的驱动程序架构，目标是简化程序开发并减少驱动中软件缺陷的数量。针对驱动程序中硬件和软件协议使用缺陷，Dingo定义了基于状态机的形式语言Tingu，Tingu支持对驱动程序行为无歧义描述。

(下转第220页)

SIGM0D. 1995:175-185

[3] Brin S, Motwai R, Ullman J D, et al. Dynamic Itemset Counting and Implication Rules for Market BasketData[C]// Proc. of ACM SIGMOD Conference on Management of Data. 1997:265-276

[4] Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules in Large Databaes[C]// Proc. of 1994 International Conference on Very Large Databases. 1994:487-499

[5] Savasere S, Omiecinski E, Navathe S. An Efficient Algorithm for Mining Association Rules in Large Databases[C]// Proc. of 21st VLDB. 1995:432-444

[6] Dunkel B, Soparkar N. Data Organization and Access for Efficient Data Mining[C]// Proc. of 15th IEEE Intl. Conf. on Data Engineering. 1999:522-529

[7] Han J, Fu Y J. Mining Multiple-Level Association Rules in Large Database[J]. IEEE Trans. on Knowledge and Data Engineering, 1999, 11(5):798-805

[8] 丁艳辉,王洪国,高明,等. 一种基于矩阵的关联规则挖掘新算法[J]. 计算机科学,2006,33(4):188-189

[9] Shenoy P, Haritsa J R, Sudarshan S, et al. Turbo-charging vertical mining of large databases[C]// Proc. of Intl. Conf. Management of Data. 2000:22-23

[10] Burdick D, Calimlim M, Gehrke J. MAFIA: a maximal frequent itemset algorithm for transactional databases[C]// Proc. of Intl. Conf. on Data Engineering. 2001:443-452

[11] Ayres J, Gehrke J E, Yiu T, et al. Sequential pattern mining using bitmaps representation[C]// Proc. of SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2002:429-435

[12] 宋长新,马克等. 改进的 Eclat 数据挖掘算法的研究[J]. 微计算机信息,2008,24(24):92-94

[13] 耿晓斐. 关联规则中 ECLAT 算法的研究与应用[D]. 重庆:重新大学,2009

[14] Zaki M, Parthasarathy S, Ogihara M, et al. New Algorithms for Fast Discovery of Association Rules[C]// Proc. of In 7th International Workshop Research Issues on Data Engineering. 1997:283-286

[15] Poovammal E, Ponnavaikko M. Utility independent privacy preserving data mining on vertically partitioned data [J]. Journal of Computer Science, 2009, 5(9):666-673

[16] Hidber C. Online association rule mining[C]// Proc. of ACM SIGMOD Intl. Conf. on Management of Data. 1999:145-156

(上接第 184 页)

描述可有效帮助驱动开发者避免错误,并可以直接生成用于验证的驱动程序属性描述。Dingo 支持直接编译驱动协议规范,运行时检查器检测驱动程序是否违反协议。

基于上述思路,该大学进一步推出 Termite[7] 系统,以实现驱动程序自动生成。主要思路是:定义操作系统接口规范和硬件接口规范;将定义好的规范形式化,生成接口协议有限自动机;将有限自动机合成为驱动程序。Termite 系统生成的 Linux 驱动程序具有同手工编写程序相同功能和接近性能。Termite 难点在于细化出程序规范,但生成的规范可复用。驱动程序自动生成技术处于试验阶段,是提高驱动程序质量的有益尝试。

结束语 随着系统对可靠性要求的提高,操作系统内核驱动程序可靠性成为国际学术界和工业界关注的热点。从改变操作系统结构的避错和容错手段,到采用程序分析验证技术直接消除驱动程序缺陷纠错手段和采用程序综合技术合成无错设备驱动,驱动程序可靠性增强是一项长期而艰巨的工作。本文从分析驱动程序缺陷入手,指出驱动程序缺陷产生的原因,并以此为线索梳理了目前的解决方法,希望对驱动程序开发和缺陷解决提供借鉴。

参 考 文 献

[1] Engler D, et al. Checking system rules using system-specific programmer-written compiler extensions[C]// Proc. of the 4th USENIX OSDI. Oct. 2000

[2] Coverity Scan Open Source Report [EB/OL]. http://www.coverity.com, 2009

[3] Arthur S. Fault resilient drivers for Longhorn server[R]. Microsoft Corporation, WinHec 2004 Presentation DW04012, May 2004

[4] Witkowski T, et al. Model Checking Concurrent Linux Device Drivers[C]// ASE'07. November, 2007

[5] Kadav A, et al. Tolerating Hardware Device Failures in Software [C]// Proc. of the 22th ACM SOSP. Dec. 2009

[6] Ryzhyk L, et al. Dingo: Taming device drivers[C]// Proc. of the 200 EuroSys Conference. Apr. 2009

[7] Ryzhyk L, et al. Automatic Device Driver Synthesis with Termite[C]// Proc. of the 22th ACM SOSP. Dec. 2009

[8] Ball T, et al. The SLAM project: Debugging system software via static analysis[C]// Proc. of the 29th POPL. 2002

[9] Ball T, et al. Thorough static analysis of device drivers[C]// Proc. of the 2006 EuroSys Conference. 2006

[10] Beyer D, et al. The software model checker BLAST Applications to software engineering[J]. Int J Softw Tools Technol Transfer, 2007

[11] Microsystems S. Solaris Express Software Developer Collection: Writing Device Drivers[M]. chapter 13. 2007

[12] Microsoft Corporation. Introduction to the WDF usermodedriverframework[EB/OL]. http://www.microsoft.com/whdc/driver/wdf/umdf intro.mspx, May 2006

[13] Hunt G, et al. Sealing OS Processes to Improve Dependability and Safety[C]// Proc. of the 2007 EuroSys Conference. 2007

[14] Tanenbaum A. Introduction to MINIX 3 [EB/OL]. http://www.osnews.com/story/15960/Introduction-to-MINIX-3/ Sep 2006 05:41 UTC

[15] Li Man-Lap, et al. Understanding the propagation of hard errors to software and implications for resilient system design[C]// ASPLOS. 2008:265-276

[16] 颜跃进,等. 操作系统设备驱动可靠性研究综述[J]. 计算机工程与科学,2009(5)