第34卷 第18期 计 算 机 工 程 2008年9月
Vol.34 No.18 Computer Engineering September 2008
·软件技术与数据库· 文章编号：1000—3428(2008)18—0093—03 文献标识码：A 中图分类号：TP393

基于内核态JVM的Linux设备驱动程序

陈 善[1]，周玲玲[1]，应忍冬[1]，戈 弋[2]
(1. 上海交通大学电子工程系，上海 200240；2. IBM中国研究院，北京 100094)

摘 要：驱动程序的不稳定是造成操作系统内核崩溃的主要原因，该文采用类型安全的Java语言开发Linux设备驱动程序以提高系统的稳定性，并分析驱动模型的结构、内核态Java虚拟机(JVM)的设计以及Java驱动程序的编写。USB网卡的测试验证了Java驱动在提高系统稳定性上的优势。

关键词：Java虚拟机；设备驱动程序；可靠性

Linux Device Driver Based on Kernel-mode JVM

CHEN Shan[1], ZHOU Ling-ling[1], YING Ren-dong[1], GE Yi[2]
(1. Dept. of Electronic Engineering, Shanghai Jiaotong University, Shanghai 200240; 2. IBM China Research Lab, Beijing 100094)

【**Abstract**】The buggy codes of device drivers usually account for most of the OS kernel crashes. This paper proposes a prototype of writing Linux device drivers in the type-safe language of Java, which effectively improves the system reliability, analyzes framework of Java device driver prototype, design of kernel-mode Java Virtual Machine(JVM) and writing of Java device driver program. USB Ethernet drivers test shows Java device drivers' advantage in improving system reliability.

【**Key words**】Java Virtual Machine(JVM); device driver; reliability

1 概述

随着计算机硬件技术的飞速发展，对操作系统的要求逐渐由“速度第一、性能第一”转移到“安全第一、稳定第一”。操作系统安全性和稳定性在许多领域得到了印证，目前流行的操作系统的稳定性不尽如人意。严格意义上说，稳定性和安全性是2个不同的概念，但从其产生的根源上看，都是因为代码中含有错误。比如，一个数组越界访问会导致系统的崩溃，使系统不稳定，而当这个越界访问被病毒或恶意代码利用时，就涉及安全性问题。本文研究目标是减少代码中错误的发生，因此，本文对稳定性及安全性的定义不严格区分。

相关研究表明[1]，设备驱动程序是引起操作系统稳定性不高的主要原因。例如，Windows XP系统85%的崩溃是由驱动引起的，而Linux内核中驱动程序的bug密度是内核其余部分的3～7倍。主要原因是：驱动程序往往是由设备厂商提供，与专业的操作系统开发者相比，他们对内核往往缺乏足够深入的了解；另外，由于硬件设备数量多、品种杂，驱动发布之前往往得不到足够的稳定性测试。由此可见，操作系统稳定性的关键在于驱动程序。如何提高驱动程序的稳定性，进而提高操作系统的稳定性，一直以来是计算机领域研究的热点。

现有研究方法按使用的技术可以分为以下几类[2]：(1)隔离保护。在驱动模块和内核之间增加隔离保护层，对驱动和内核之间的交互进行监控，当驱动发生错误时，错误被限制在驱动内部，不会扩散至内核，如Nooks。(2)微内核技术。将驱动变为独立的用户态进程，驱动中的错误最多，会导致自身进程的终结，对操作系统没有影响，如Minix3。(3)虚拟机技术。驱动和内核运行在各自独立的虚拟机中，驱动的错误最多，会导致驱动及其所在虚拟机的终止，而不会造成内核的崩溃，如L[4]Linux。(4)语言级安全。采用类型安全的语言编写驱动程序甚至整个操作系统，由编译工具在编译时保证驱动程序的安全性，如微软的Singularity。

这些方法一定程度上改进了系统的稳定性，但并不完美，存在着一定的局限性，如错误处理能力有限、与现有系统不兼容等。

本文采用Java来开发设备驱动程序，面向现有操作系统，改善因设备驱动而导致的系统稳定性问题。研究内容包括：(1)Java驱动模型的结构。(2)Java运行平台——内核态Java虚拟机的设计。(3)Java驱动程序的开发。

2 Java驱动的安全性分析

Java是一种类型安全的语言，Java驱动由Java虚拟机解释执行，Java虚拟机对Java驱动进行了一定的隔离保护，因此，Java驱动可以说是对现有方法的借鉴和综合。Java驱动的安全性主要来源于：

(1)编译时安全

1)Java是一种类型安全的语言，不支持指针操作，消除了C语言中很多安全问题的根源。

2)Java程序编译时，Java编译器会对Java程序进行严格的安全检查，如错误的或不安全的类型转换，最大程度地保证代码的安全性。相比于C语言，Java牺牲了一些灵活性，但换来的是安全性的提高。

(2)运行时安全

Java是一种解释执行语言，每一条Java指令的执行都在JVM掌控之下，JVM内建立的安全机制会对Java指令进行严格的检查，当检查到有错误如数组越界访问、空对象引用

作者简介：陈 善(1982－)，男，硕士研究生，主研方向：嵌入式系统；周玲玲，副教授；应忍冬、戈 弋，博士
收稿日期：2007-11-28 **E-mail：**chenshan_xp@sjtu.edu.cn

访问时，可以中止指令的执行，同时抛出相应异常请求处理。若异常得到正确处理，则程序可以继续运行；若 JVM 无法处理该异常，则 JVM 选择安全退出，以防止错误对系统的破坏，保证了安全性。

值得一提的是，与一般研究方法相比，Java 驱动除了具有高安全性之外，还具有如下优势：

(1)易于调试。Java 的解释执行特性决定了它易于调试。提供内核态的 jdb 调试工具是一个办法，除此之外，还可以借助 JVM 的异常机制。如上所述，当错误发生时，JVM 会抛出异常，此时可以输出异常的相关信息，如异常类型、异常位置等，有了这些信息，再辅助以某些字节码分析工具如 Jad，就可以迅速地定位程序中的错误，加快开发效率。这有效地解决了内核态调试困难的矛盾。

(2)面向对象。Java 不但是一种类型安全语言，更是一种彻底的面向对象的语言，因此，用 Java 来开发驱动，可以充分利用面向对象的编程思想，提高开发效率。

3 模型架构

Linux 内核中的设备驱动见图 1。在 Linux 内核中，设备驱动常常以独立模块的形式存在，驱动的作用是为内核屏蔽复杂的设备操作细节。当内核要使用访问外设时，只需调用驱动提供的标准化接口，而无须关心驱动内部的具体实现。Java 驱动模型正是基于这一点，即驱动向上提供的接口未变，改变的只是内部实现，从而与内核之间保持了良好的兼容性。

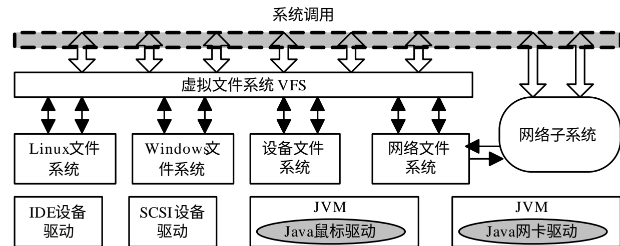


图 1 Linux 内核中的设备驱动

Java 驱动程序由内核态 JVM 解释执行，内核态 JVM 是 Java 驱动模型的关键。Java 驱动模型的内部结构见图 2。

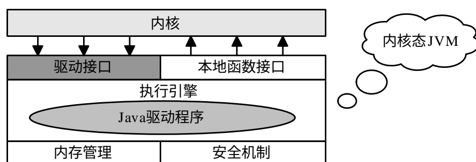


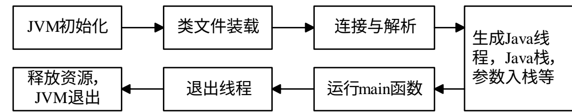
图 2 Java 驱动模型

内核态JVM与用户态JVM[3]在内部结构上基本相同，都有执行引擎、内存管理、安全机制以及本地函数接口等，不同之处在于：内核态JVM增加了驱动接口。驱动接口的主要任务是将内核传递的参数封装成Java驱动中的参数形式，然后调用相应的Java函数完成对具体外设的操作，Java函数执行完成后，驱动接口获取返回值返回给内核，一次外设操作完成。Java驱动程序的功能是各种设备操作的具体实现。Java驱动程序的编写必须遵守一定的规范，为各种类型的设备(如字符型设备、块设备、USB设备)定义了各自的接口，Java驱动程序必须实现相应的接口。

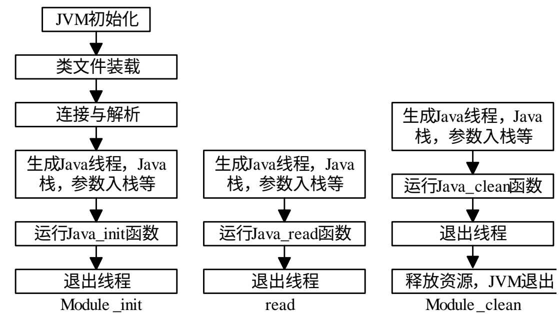
4 设计与实现

4.1 执行流程

用户态 JVM 运行的是应用程序，内核态 JVM 运行的是驱动程序。应用程序与驱动程序的不同之处在于：应用程序完成一项任务时，一般以进程的形式存在；而驱动程序的功能是向上提供操作设备的统一接口，一般以模块化函数库的形式存在，供内核调用。这就造成了用户态 JVM 和内核态 JVM 在执行流程上的差异，如图 3 所示。



(a)用户态 JVM



(b)内核态 JVM

图 3 JVM 的执行流程

在内核态 JVM 中，驱动接口中的 Module_init 和 Module_clean 2 个函数分别在驱动模块被装载和卸载时调用，完成驱动的初始化和清理工作，因此，它们是必需的；read 函数则代表了驱动接口函数的一般执行流程，具有通用性。由此可见，驱动接口具有较好的设备无关性，即驱动接口只完成对 Java 驱动函数的调用，本身不进行任何设备相关的操作，这使得驱动易于升级且维护简单。

4.2 驱动接口

4.2.1 实现描述

在接口的实现中，内核态 JVM 的实现是基于开源 JVM (TinyVM)的，该 JVM 直接运行在一个不带操作系统的微控制器上。下面以字符型设备驱动接口中的 read 函数为例，给出驱动接口函数中主要步骤的代码级描述：

(1)参数封装

```
jfile＝new_object (JAVA_DRIVER_JFILE);
map_para(jfile, filep);/*filep 是内核传递的文件描述符参数，它被封装进 Java 对象 jfile 以供 Java 函数使用*/
```

(2)生成 Java 线程

```
jthd=new_object (JAVA_LANG_THREAD);
init_thread(jthd); /*线程初始化，包括为 Java 线程生成 Java 栈，初始化字节码指针 pc，栈指针 sp，局部变量区基地址指针 lp */
enqueue_thread(jthd);/*驱动中同样存在多线程，比如一个读线程，一个写线程，因此使用一个线程队列进行管理*/
```

(3)参数传递

```
push_word(jthd, jfile);
/*Java 函数调用的参数也是通过 Java 栈传递的*/
```

(4)运行 read 函数

```
mRec=find_method(READ);
engine(); /*启动执行引擎*/
```

(5)结束退出

```
dequeue_thread(jthd);
return ret;
```

4.2.2 参数的传递

内核调用驱动接口时传递的参数往往是内核定义的结构体的指针，Java 无法直接访问，因此，驱动接口需要将其转化为 Java 支持的数据类型。每个内核结构体用一个特定的

Java 类的对象来封装，Java 类由一个私有数据域及若干公有函数组成。私有数据域保存结构体地址，公有函数则实现对结构体成员的各种操作。这样虽然对象内有结构体地址，但 Java 驱动只能通过调用类的成员函数实现对结构体的访问，而不能直接操作结构体地址，从而维护了 Java 语言的类型安全特性。另外，通过函数访问内核结构体的方式可以对 Java 驱动对内核结构体的访问权限进行合理的限制，只有被允许的访问才提供相应的函数，同时在函数内部还可以进行相关的安全性检验，体现了 Java 驱动的安全性目标。

4.3 I/O 操作与中断处理

I/O 操作是驱动程序各项功能得以实现的基础。I/O 操作常常包含了对 I/O 端口地址或内存映射的 I/O 地址的读写，由于 Java 不支持指针，因此它无法实现对 I/O 设备的直接访问。通过本地函数调用间接实现，地址以参数的形式传递给本地函数，本地函数以此地址直接读写 I/O 设备或调用操作系统的底层硬件接口，最终完成 I/O 操作。这是一种变相的指针操作，有可能破坏 Java 语言的安全特性，因此需要在本地函数中进行有关的安全性检验，如地址范围、访问权限等，以尽量维护 Java 的安全性。

在 Linux 中，中断处理程序与一般函数的不同之处在于：运行时没有进程上下文，因此，不能进行引发睡眠的操作。中断处理程序睡眠错误是 C 驱动中导致内核崩溃的常见原因之一。Java 驱动也可以避免该类错误的发生。具体做法为在可能会引发睡眠的操作之前检测所处运行环境，判断是否为中断上下文。

接下来又有 2 种情况：(1)若所要进行的操作有中断与非中断两种实现，则选择相应的实现，程序继续运行。如创建新对象时，空闲内存不足需要向系统追加内存，若判断在中断处理中，kmalloc()则使用 GFP_ATMIC 标志，若不是，则可以使用 GFP_KERNEL 标志。(2)若所要进行的操作只能在非中断处理程序中使用，则抛出相应异常请求处理。若得到处理则程序继续运行，否则终止运行。异常处理函数可以由系统定义，或者由驱动程序自定义，后者被调用的优先级高于前者。

Java 驱动对中断睡眠错误的处理展示了 Java 驱动在错误处理上的灵活性和可扩展性，通过这种形式，可以不断丰富和完善 Java 驱动的安全特性，使其能够发现并处理更多类型的错误。

5 Java 驱动程序的编写

Java 驱动程序的设计采用面向对象的编程思想，驱动程序被抽象成类，其中的成员函数向外提供了访问该类设备的统一接口，数据域则存放设备相关的数据，如设备状态，配置等。每一个类的对象表示一个具体物理设备。Java 驱动程序见图 4。

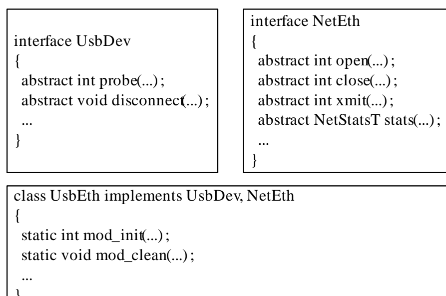


图 4 Java 驱动程序

6 示例测试

用 Java 实现 Linux 源码树中自带的 USB 鼠标驱动(drivers/usb/usbmouse.c)和 USB 网卡驱动(drivers/usb/pegasus.c)，由于鼠标驱动较为简单，因此测试主要在网卡上进行。测试平台配置为 Intel P4 1.5 GHz，内存 512 MB， OS 为 RedFlag4.0，内核版本 2.4.20。

6.1 系统稳定性测试

(1)测试方法

在 C 和 Java 中引入相同类型的错误，然后分别测试其对系统稳定性的影响。C 语言中常见的运行时错误类型主要有内存访问错误，包括数组越界和空指针等，错误类型转换，如函数调用时参数类型不兼容，以及算术运算中的除 0 错误。Java 有对象引用，没有指针，对应的错误类型有：数组越界，空引用访问，引用类型错误及除 0。

(2)测试结果

C 驱动和 Java 驱动的测试结果分别如表 1 和表 2 所示，结果的严重程度由高到低共分 5 种类型：1)错误未被发现，可能破坏内核数据；2)内核崩溃；3)内核报错，输出 Oops 消息；4)错误被检测到，抛出异常请求处理，若无法处理则 JVM 安全退出；5)编译时错误。

表 1 C 驱动下的系统稳定性

错误类型	进程上下文	中断上下文
数组越界	1)	1)
空指针	3)	2)
错误类型转换	1)	1)
除 0	3)	2)

表 2 Java 驱动下的系统稳定性

错误类型	进程上下文	中断上下文
数组越界	4)	4)
空引用	4)	4)
引用类型错误	5)	5)
除 0	4)	4)

测试结果表明，Java 驱动能有效地降低驱动错误对操作系统的影响，提高系统的稳定性。在 C 驱动中，运行错误很容易导致内核的崩溃甚至会破坏内核数据，而在 Java 驱动中，运行错误均被 JVM 捕获，并抛出相应异常。由于没有实现相应的异常处理程序，因此 JVM 将异常信息输出至屏幕后安全退出，避免了继续运行可能对内核数据造成的破坏，从而保护了内核。

6.2 性能测试

局域网传输一个大小为 20 MB 的文件，C 驱动与 Java 驱动性能测试的比较见表 3。其中，USB 网卡使用的是 USB1.1 规范。Java 驱动与 C 驱动的传输速度比值为 1.00；CPU 的占用率比值为 1.53。

表 3 C 驱动与 Java 驱动性能测试的比较

驱动	最大传输速度/(KB·s^{-1})	丢包或错误包	CPU 占用率/(%)
C	440	0	3.4
Java	440	0	5.2

测试结果显示：两者传输速度相同，且均未发生丢包或错误包，也就是说网卡性能未受影响；对于 CPU 占用率，相比较而言，Java 驱动比 C 驱动相对多使用了 53% 的 CPU 资源，但由于 CPU 的绝对占用率较低，因此对系统整体性能的影响也较小。通常来说，Java 的执行效率要比 C 低很多，而此处代表执行效率的 CPU 占用率，Java 驱动和 C 驱动相比并没有理论值那么高，主要是因为 Java 驱动和 C 驱动一样，

(下转第 98 页)