**理解现代设备驱动程序**

**陈双-P17206032-中电15所**

**介绍：**

设备驱动程序是操作系统内核代码的最大贡献者，Linux内核中有超过500万行代码，并导致显著的复杂性、错误和开发成本。近年来，一系列研究旨在提高驱动的可靠性，简化驱动的发展。然而，对于这一庞大的代码体，除了为研究所用的一小部分驱动程序之外，其他都是未知的。

在本文中，研究了Linux驱动程序的源代码，了解驱动程序实际做什么，当前研究如何应用于他们，以及未来研究的机会。我们确定驱动研究所做的假设是否真的是真的。我们还分析驱动程序代码和抽象，以确定驱动程序是否可以受益于代码重新组织或硬件趋势。我们开发了一套静态分析工具来分析不同轴上的驱动程序代码。

**背景原因：**

我们发现，许多驱动研究所做的假设并不适用于所有驱动。至少44%的驱动程序具有未被类定义捕获的代码，28%的驱动程序支持每个驱动程序多于一个设备，并且15%的驱动程序对数据进行显著的计算。从驱动程序交互研究中，我们发现USB总线提供了一个高效的总线接口，具有显著的标准化代码和粗粒度访问，理想地用于隔离地执行驱动程序。我们还发现，对于不同的总线和类的驱动程序具有广泛的不同级别的设备交互，这表明隔离的成本将根据类别而变化。最后，从我们的驱动程序相似性研究，我们发现8%的所有驱动程序代码基本上类似于其他地方的代码，可以删除新的抽象或库。

然而，对设备驱动程序的研究大多集中在设备的一小部分，通常是网络卡、声卡和存储设备，所有这些都使用PCI总线。这些只是所有驱动程序的一小部分，并且这些设备的结果可能不推广到全套驱动程序。例如，许多消费PC的设备通过USB连接。类似地，所研究的设备相当成熟并具有标准化接口，但许多其他设备可能具有显著的功能差异。

因此，重要的是研究所有的驱动程序，以审查如何驱动研究的解决方案是适用于所有类别的驱动程序。此外，更好地理解驱动程序代码可以导致新的抽象和接口，可以减少驱动程序的复杂性和提高可靠性。

**文章贡献：**

本文的贡献如下：首先，我们分析驱动程序代码是为了验证驱动程序研究所驱动的代码的共同假设。我们表明，虽然这些假设持有大多数驱动，有大量的驱动违反这些假设。我们还发现，一些快速增长的驱动程序类没有被驱动研究解决。其次，我们研究与内核和设备的驱动程序交互，以发现现有驱动架构如何适应多核处理器、高功率处理器和虚拟化I/O设备的世界，我们发现驱动程序变化很大。然而，进行TCP卸载的网卡可能具有仅在设备中可用的重要协议状态，并且不能通过监视内核/驱动程序接口来捕获。

虽然这些假设都适用于许多驱动，这项研究试图量化他们真正的普遍性。如果这些假设对于所有的驱动都是正确的，那么这些研究思路具有广泛的适用性。如果没有，则需要新的研究来解决离群点。

**具体实现：**

我们在整个Linux驱动程序源上运行这些分析，并将输出存储在SQL数据库中。数据库存储每个驱动程序的信息以及驱动程序中的每个函数。有关驱动程序的信息包括名称、路径、大小、类、芯片组数、模块参数和与内核注册的接口。每个驱动函数的信息包括函数名、大小、标签、分配的资源（内存、锁等），以及它如何与内核和设备交互。从数据库中，确定专用于任何函数的代码量是一个简单的查询。

驱动程序在不同用途中使用的代码差别很大；单个芯片组的简单驱动程序可以将其大部分代码用于处理I/O请求，并具有简单的初始化例程。相比之下，支持几十个芯片组的复杂驱动程序可能有更多的代码用于初始化和错误处理，而不是请求处理。

通过结果表明，虽然关于驱动程序的共同假设一般是正确的，但是由于驱动程序的多样性，人们不能假定它们总是持有。特别是，许多驱动程序包含大量的代码，使得一些现有的研究，如驱动器的自动生成困难。

前面的部分集中在驱动代码的功能上，这里我们转向驱动代码的交互：驱动程序是如何使用内核的，以及驱动程序是如何与设备通信的？我们看到三个原因来研究这些相互作用。首先，主机上的设备或额外内核上的额外处理能力提供了重新设计驱动架构以提高可靠性和性能的机会。例如，可以将许多驱动程序函数移出内核并进入设备本身。或者，在虚拟化系统中，驱动程序功能可以在不同虚拟机上的不同内核上执行。第二，在操作系统之间移动驱动程序的困难很大程度上来自驱动程序/内核接口，因此研究内核驱动程序的请求有助于设计更多的便携式驱动程序。第三，隔离和可靠性的成本与接口的大小和交互的频率成正比，因此理解接口可以导致更有效的容错机制。

支持基础设施很少的驱动程序显示了与内核的频繁交互以访问内核服务，但很少调用设备支持代码。具有高抽象级别的驱动程序对内核的调用很少。具有支持库的驱动程序演示了对内核泛型例程的频繁调用以及对设备支持例程的调用。

很少调用设备库的驱动程序可能具有低抽象级别，因此是提取公共功能的候选对象。类似地，具有多个内核交互和设备库交互的驱动程序可能会受益于转换成分层的“迷你端口”架构，其中更多的驱动程序功能被提取到一个公共库中。

本研究的目的是研究Linux中的完整驱动程序集，以避免从通常用于研究的一组驱动程序中推广，并形成新的泛化。将更多的驱动程序功能移出内核，甚至在设备本身上。此外，许多驱动程序只需要很少的硬件访问，而几乎完全与总线交互。因此，这样的驱动程序可以有效地运行而不需要特权，因为它们不需要特殊的硬件访问。我们发现，USB和NeXBUS提供了机会，利用额外的周期在设备上执行驱动器上的他们，可以有效地用于去除驱动程序从内核只留下标准化总线代码在内核中。

**小结：**

证据表明，有大量的机会来减少驱动代码的数量。相似性分析表明，有许多类似的代码模式的实例，可以用更好的库抽象来代替，或者在某些情况下可以用表代替。虽然这项研究是针对Linux进行的，但我们相信许多类似的模式，虽然对于不同类别的设备，将显示在其他操作系统中。比较驱动代码跨操作系统的差异也可能是有趣的，这可能会显示效率或复杂性的细微差别。