微内核通用：宏孟生产微粒的性能及兼容性

陈海波、华为中央软件研究所、上海交通大学等教授；
谢苗、宁佳、王楠、于丽、刘年、刘玉涛、王飞飞、黄强、
李坤、阳洪阳、王慧、殷杰、于鹏、徐凤伟，
华为中央软件研究所
https://www.美国尼克斯。组织，会议，演讲，演讲

这篇论文包括在论文程序中
第18届USENIX操作系统研讨会
设计和实施。
2024年7月10日至12日，美国加州圣克拉拉
978-1-939133-40-3

开放获取有关机构的法律程序
第十八届USENIX运营研讨会
系统的设计与实现
是由

微内核概述：宏孟生产微内核的性能和兼容性

陈海波1、2、谢苗[1]、宁家[1]王南[1]李彧[1]刘年[1]刘玉涛[1]王飞[1]黄强[1]李昆[1]杨宏阳[1]王晖[1]、洁阴[1]、于鹏[1]和徐凤伟[1]

[1]*华为中央软件研究所，*[2]*上海交通大学*

摘要

安全性、可靠性和可扩展性的优点使得最先进的微内核在嵌入式和安全关键场景中普遍存在。然而，当针对智能手机和智能汽车等更一般的场景时，它们将面临着性能和兼容性问题。

本文介绍了宏孟核（HM）的设计和实现，这是一种商业化的通用微内核，在解决上述挑战的同时保留了微内核的大部分优点。出于商业实用性的考虑，我们将HM设计为与Linux API和ABI兼容，以重用其丰富的应用程序和驱动程序生态系统。为了使它在兼容性和通用性的限制下保持性能，我们重新检查了传统的微内核智慧，包括IPC、基于能力的访问控制和用户空间分页，并相应地修改它们。具体来说，我们认为每个调用IPC不是性能的唯一问题，而是IPC频率、操作系统服务之间的状态双重簿记以及隐藏内核对象的功能都会导致显著的性能下降。我们通过一组技术相应地减轻它们，包括差异化的隔离类、灵活的组合、无策略的内核寻呼和基于地址令牌的访问控制。

HM由一个最小的核心内核和一组最特权的操作系统服务组成，它可以运行像AOSP和Open和谐这样的复杂框架。HM已在新兴场景中部署在数千万台设备上，包括智能路由器、智能汽车和智能手机，通常与Linux设备相比，性能和安全性都有所提高。

1介绍

微内核将内核中的功能最小化，并将组件，如文件系统和设备驱动程序，移动到焊接化和特权最低的操作系统服务中，实现了比单片内核更好的可靠性、安全性和可扩展性比如Linux。由于这些优点，最先进的（SOTA）微内核已经被广泛地部署在嵌入式和安全关键场景[30,52,54]中。

另一方面，虽然像Linux这样的单片内核在服务器和云等通用场景中占主导地位，但越来越多的新兴场景，如智能汽车和智能手机，除了良好的性能外，还需要更好的安全性、可靠性和可扩展性，而Linux不太适合。虽然是通用的，但Linux更多地转向服务器和云，这使得其他场景不那么有益。例如，先发制人-rt补丁[1]花了10多年的时间才部分合并，它的发展仍然不是主流，更不用说其他特定领域的策略[20,21]了。此外，Linux注定要难以（如果可能的话）满足此类场景[98,113]所需的高级行业认证。

然而，尽管微粒已经被广泛研究了几十年[16、28、30、46、49、52、52-54、64、67、73、75、76、86]，但SOTA微粒主要针对一些特定的结构域，e。g.，嵌入式和安全关键的。它们通常使用静态资源分区和分配，并且缺乏通用的操作系统功能来运行商业现成的应用程序。下面，我们总结了为这种新出现的场景将微内核作为通用操作系统内核进行改造的主要挑战。

*兼容性：兼容POSIX子集是不够的。*重建整个软件生态系统是不切实际的。因此，SOTA微内核，如seL4 [67]和Zircon [46]，通过提供自定义库，实现了最小的POSIX子集遵从性。g.，musl-libc [47]，它生成对操作系统服务的进程间调用（IPC）。然而，它们面临着[6,116]的部署问题，例如不能二进制兼容，以及在新兴的场景中实现挑战，例如分叉和轮询。此外，它们很难重用具有负担得起的工程工作量和不妥协的性能的设备驱动程序，而这对生产部署至关重要。

*性能：IPC并不是唯一值得关注的问题。*在新兴的场景中，性能是首要任务，它直接决定了用户体验。而SOTA的微粒则像seL4 [67]一样

最近的架构支持[28,49,86]已经实现了创纪录的高IPC性能，我们观察到它们仍然会造成巨大的性能开销，因为当微内核普及时，IPC频率会显著增加（智能手机比路由器高70倍）。此外，我们还观察到由于多服务器设计，状态双重簿记导致的同样严重的性能问题，这引入了额外的性能开销（比Linux慢2倍）和内存占用（35%）。此外，基于功能的访问控制，它将频繁更新的内核对象隐藏在功能后面，由于频繁的调用，可能会导致巨大的开销。例如，它会导致页面故障处理是
比Linux慢3.4倍。

我们在7年前开始了宏孟内核（HM）项目，重新检查并将微内核改造为一个通用的操作系统内核。为了便于生产部署，HM实现了完整的Linux API/ABI兼容性，并能够重用Linux应用程序和驱动程序生态系统，这样它就可以运行复杂的框架，如AOSP [42]和Open和谐[35]与丰富的外设。尽管兼容性目标可能会限制其性能，但HM仍然将性能作为其主要重点。因此，HM尊重微内核的设计原则，但没有经过谨慎的妥协。具体来说，HM做出了以下关键的设计决策。

最小的微内核和最少的操作系统服务。HM只保留了核心内核中必要的功能，包括线程调度程序、串行/计时器驱动程序和访问控制，并保留了所有其他组件作为核心内核之外的隔离操作系统服务（多服务器）。此外，HM采用细粒度访问控制，以保持最小特权原则，以更好的安全性。因此，HM继承了微内核的安全性和可靠性优势。

通过实现LinuxAPI/ABI兼容性和具有高性能要求的驱动程序重用来最大化兼容性。HM通过兼容ABI的shim实现完整的Linux API/ABI兼容性来集成现有的软件生态系统，以识别和重定向Linux系统尺度到IPCs。此外，HM通过一个驱动程序容器重用未修改的Linux驱动程序，该驱动程序容器为HM提供Linux运行时的工程工作，并通过使用双驱动程序分离控制平面和数据平面，消除了关键的路径性能下降。

性能首先由结构支撑件完成。HM在不违反微内核的架构原则的情况下优先考虑性能。具体来说，HM实现了灵活的组合，以分层地放松可信服务之间的隔离，以最小化IPC开销，并合并紧密耦合的服务，以最小化IPC频率，并在性能要求很高的场景中消除状态双重簿记，同时保持在安全关键场景中分离它们的能力。HM还补充了具有性能的能力
基于令牌的访问控制，促进了有效的合作，如无策略的内核分页。

我们已经在数千万台设备上部署了HM，包括智能路由器、智能汽车和智能手机，这不仅提供了更好的安全性和可靠性，而且比Linux提供更好的性能。HM的关键组件是半正式验证的[55]，通过正式指定设计和使用自动验证和验证引导的测试来验证关键的安全属性，如无整数和缓冲区溢出。HM已通过ASIL-D[61]（安全）和CCEAL6+[62]（安全）认证。在路由器中，HM通过减少30%的系统内存占用，允许增加30%的客户端连接。在车辆中，HM实现了60%的启动时间和60%的跨域延迟。在智能手机中，HM将应用启动时间缩短了17%，帧下降减少了10%。

2.对于一般的微内核的情况

.12微核审查

微内核的一个主要标志是最小化原则[73,76]，它将核心内核中的功能最小化，并将其他函数转移到用户空间服务中。SOTA微内核还采用了基于能力的细粒度访问控制[46,52,67,74]，以保持最小特权原则。因此，微内核本质上比单片内核[12,79]更安全、可靠和可扩展。

然而，尽管微粒已经被广泛研究了几十年[163052-546467737576122]，SOTA微粒主要针对特定的领域，如嵌入式和安全关键系统。例如嵌入高通蜂窝调制解调器芯片[30]，QNX的L4[1]在汽车和嵌入式系统[54]，和Zircon（紫红色的内核）在智能音箱[46]。关于微内核如何被扩展为智能汽车和智能手机等新兴场景的通用操作系统内核的研究还很少。

该行业采用了混合内核，如[88][88]和苹果XNU [4]，它们结合了一个核心微内核，e。g.，XNU中的Mach，以及内核空间中的所有其他服务（作为一个整体），例如，NT中的执行服务和XNU中的BSD服务。尽管混合内核也最小化了核心内核中的功能，但它们并没有继承微内核的许多优点。例如，混合内核中的操作系统服务不是特权，也不是很好地隔离。因此，任何受损或有问题的操作系统服务都可能损坏系统[88]，并可能导致严重的后果，例如损坏用户数据。

.22.对一般微内核的需求

智能汽车和智能手机等新兴场景需要丰富的外设和应用程序。例如，汽车的行业标准已经发展到需要更丰富的操作系统

[1]虽然QNX曾经支持平板电脑/手机[14]，并通过虚拟机运行AOSP应用程序，但由于兼容性和性能有限，QNX停止了这一功能，并已完全过渡到嵌入式市场[13]。

功能[7]。与此同时，新兴的场景也强调安全性和安全性。例如，车辆需要对乘客安全的高可靠性，而智能手机则需要增强的安全性来保护敏感数据。我们将在下面列出与特定领域场景的主要区别。

软件生态系统。*在特定于领域的场景中，应用程序大多是定制的和源代码可用的。*因此，兼容posix被认为足以进行应用程序移植（基于我们的部署经验甚至不是如此）。然而，在智能手机等新兴场景中，应用程序和库通常以二元形式分布，而框架需要的不仅仅是POSIX遵从性[6]，后者要求Linux ABI兼容性。

资源管理。在特定于领域的场景中，只有少数预先确定的应用程序，而且硬件资源也很有限。因此，应用程序主要是自己管理资源，而内核主要负责保留资源。然而，在新兴的场景中，相互竞争的应用程序需要协调的资源管理。内核需要更成熟的功能，如有效的资源管理和公平的分配。 **表演**在特定于领域的场景中，微内核优先考虑大多数静态应用程序的安全性和严格的资源（例如，时间）隔离，在这些应用程序中，性能不是主要问题。然而，在新兴的场景中，性能也是首要任务，它直接决定了用户体验，从而决定了内核的广泛部署。

人们要求同时集成丰富的软件生态系统和功能，以及安全性和可靠性，这使得现有的操作系统很难同时满足它们。一种方法是为这样的场景定制像Linux这样的库存操作系统，比如Linux，不幸的是，随着上游的发展非常昂贵（第2.3节）。之前的工作还提出了各种架构，包括单核[65,81,102]、多核[9]、外核[31]和分核[109]。然而，它们主要针对具有明确资源分离的服务器场景，同时缺乏对新兴场景中所需的高效和协调的资源管理的支持。此外，分割状态所带来的同步开销和复杂性使实现兼容性具有挑战性。

因此，我们认为有必要探索另一种将微内核演化为通用操作系统内核的途径。

.3Linux的2个问题

Linux已经主导了服务器和云市场，并处于-越来越多地渗透到其他领域，如PC和嵌入式领域。然而，它是以牺牲安全性、可靠性和性能为代价的，特别是在新兴的场景中。 **安全和可靠性。**Linux模块，如文件系统（FS）和设备驱动程序，覆盖了其3000万行代码库的80%。它们造成了大多数缺陷和漏洞（占过去4年总共1000个CVE [23]的90%），并显著降低了可靠性和安全性[19]。此外，大约80%的cve是数据泄漏，可以通过适当的隔离来避免。因此，一系列长期的研究[18、25、38、48、56、83、90-92、100、105、106、112、120、123]的目的是以分隔的方式将内核从模块中分离出来。然而，固有的紧密耦合需要大量的工程工作，甚至需要重写[56,90,91]。此外，内核模块api和安全补丁的不稳定性迫使它们频繁升级，使得它们在现实部署中不太实用。

通用性vs。特化虽然Linux针对的是一般场景，但最近的补丁和特性表明，创新主要是由服务器和云驱动的，这甚至阻碍了其他场景[89,103]的性能。此外，具有丰富外设和各种场景的设备的多样性越来越多，需要专门的策略来利用性能和能源效率的净空间，如根据服务质量[20,21]分配资源或最小化空间使用[119]。然而，由于内核模块固有的紧密耦合，这种策略需要大量的工程工作来定制内核。虽然[58,66,78,84,93]付出了很多努力来提高可定制性，但很难将它们集成到主流内核中。

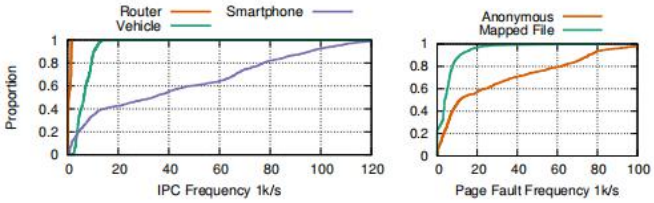
定制vs。演化另一个问题是如何开发自定义代码。与上游同步需要大量的努力来重新应用更改，而不同步可能会使系统面临安全漏洞。多年的生产经验表明，由于内核内部api的频繁变化，这是非常昂贵的，而性能回归需要大量的努力来定位甚至重新设计整个补丁。这严重限制了现实世界部署中的可定制性。因此，市场上的大量产品仍在运行Linux 2.6 [50,51,117]，它在7年前达到生命终结（EOL），有许多已知的安全漏洞[24,117]。

3重新审视微内核

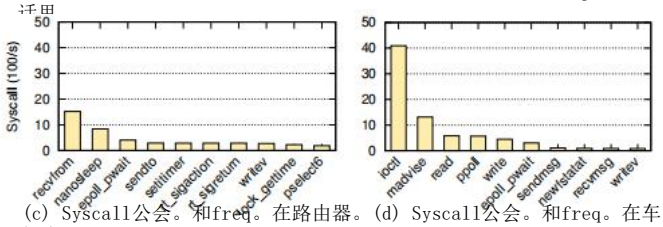
.13缩放微核

在新兴场景中部署微内核在性能和兼容性方面都带来了挑战。图1展示了在生产中部署HM所观察到的新兴场景的特征。对于路由器，我们直接从生产环境中收集数据。对于车辆和手机，我们重播了一个典型的使用（持续24小时），这些使用来自记录的大量大规模执行（匿名和用户同意）。

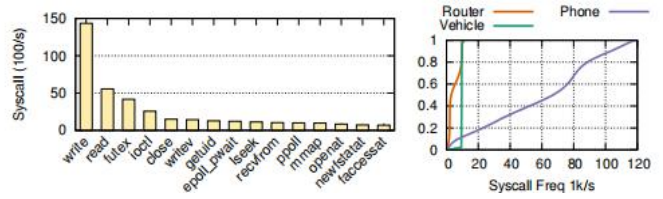
观察1：在新出现的情况下，IPC频率迅速增加。图1a显示了在配置所有操作S服务隔离在用户空间时HM中的IPC频率CDF。智能手机(平均。41k/s)和车辆（7k/s）的IPC频率远高于路由器（0.6k/s，更类似于特定领域的场景）。图1b、1e和1f通过显示次要的（即，不是来自磁盘/设备）来说明这一点



(a) IPC自由度。CDF。用户空间中的所有服务。(b)页面故障freq。在电话里



(c) Syscall公会。和freq。在路由器。(d) Syscall公会。和freq。在车辆中



页面故障的频率，以及电话中系统尺度的分布和频率。如图所示，高IPC频率不仅是由于较高的系统通频率（61k/s，比路由器高13倍），而且通过调用大量的文件操作（IPC到FS），并触发大量内存映射文件故障（5k/s），这需要内存管理器和FS之间额外的IPC往返。因此，我们不仅应该优化IPC的性能，而且还应该最小化IPC的频率。

观察2：分布式多服务器导致状态双重簿记。最小化原则确定了对于共享对象没有集中的存储库，例如文件描述符（fd）和页面缓存，并将它们分布在多个地方。然而，如图1c1e所示，新兴场景中的应用程序经常调用像轮询这样的函数，它依赖于对此类状态的集中管理。图2进一步展示了应用程序启动的CPU火焰图，它在很大程度上依赖于文件映射的性能，对用户体验[45]至关重要。如图所示，16%的时间花在处理页面缓存丢失上，这就引入了额外的IPC往返，比Linux慢2倍。此外，在智能手机中120MB的基础上（FS+mem）之上，页面缓存的双重簿记消耗了额外的50MB内存。

观察3：能力抑制了有效的合作。功能是隐藏内核对象背后的功能，由于频繁更新一些内核对象（例如页面表），会导致显著的性能开销

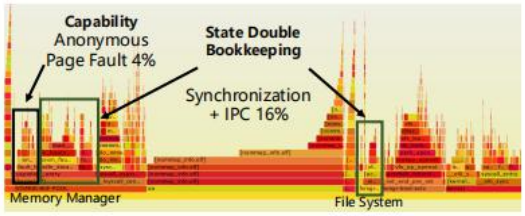


图2：HM中智能手机应用程序启动的CPU火焰图。服务合并和内核分页将被禁用。

在内核之外进行管理，并抑制它们之间的有效合作。例如，这可能会导致匿名页面错误的处理比Linux慢3.4倍，这经常发生在智能手机(平均。27k/s，图1b中80%的小页面错误)，并增加了应用程序启动时间的巨大开销（图2中的4%）。

观察结果4：生态兼容性要求的不仅仅是符合POSIX的要求。许多SOTA微内核通过提供直接链接到应用程序并生成IPC到OS服务的自定义运行时库[47]，实现了POSIX遵从性的最小子集。然而，它面临着不兼容二进制兼容和需要定制的构建环境的部署问题。此外，由于Linux使用文件作为统一接口，而微内核中已不再存在，因此实现高效的fd复用和ioctl等向量系统尺度也具有挑战性，这在新兴场景中经常使用，如图1c-1e所示。

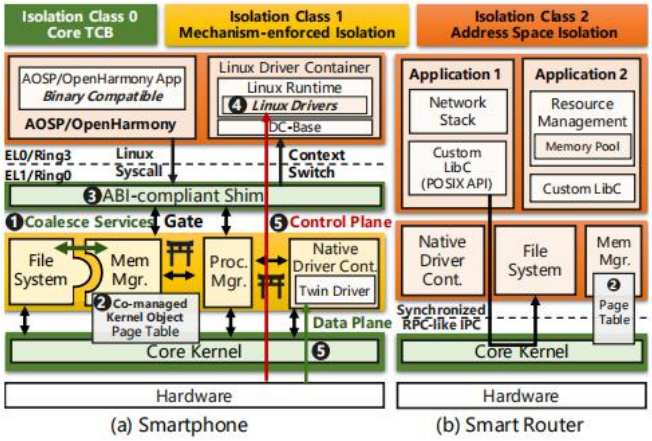
观察5：在新兴场景中的部署需要有效的驱动程序重用。当在智能手机上部署HM时，我们观察到正常运行所需的驱动程序数量大量增加。对于路由器，需要的司机数量不到20人（主要是内部维护），而车辆和手机的司机数量将增加到超过700人。我们的估计表明，重写这些驱动因素需要超过5000人年，而要想获得成熟并保持发展还需要时间。因此，重用设备驱动程序是一个更合理的选择。然而，以前的工作，包括移植驱动程序[3,17,32,41,118]的运行时环境和使用虚拟机[72]，面临着兼容性、工程工作和性能挑战（在第5.2节中讨论）。

.23鸿蒙概况

HM尊重微内核的核心设计原则，但没有达到极端程度，以谨慎地解决新兴场景中的性能和兼容性挑战。我们在表1中总结了HM的设计决策，并在下面列出了设计原则。图3显示了HM的概述。 **原则1：保留最小值。**微内核的安全性、可靠性和可扩展性源于三个基本的架构设计原则，包括分离策略和机制、解耦和隔离操作系统服务，以及强制执行细粒度的访问控制。混合内核还通过代码解耦但没有适当的隔离来实现最小化。因此，它没有继承mi-的主要好处

表1：红孟的设计决策。

	SOTA微粒	混合核	洪孟的设计
最小性	最小核	代码解耦	保留的：具有孤立的、特权最小的操作系统服务的最小微内核。
内部过程通信	带快速路径的IPC	函数调用	增强型：同步RPC地址资源alloc./exhaustion/acct。流出
隔离	用户空间服务	带内核的合并	灵活化：针对定制的隔离和性能的差异化隔离类别。
组成	静态多服务器	静态单服务器	**灵活的组合：灵活的组合，以适应不同的场景。**
访问控制	基于能力	对象管理器	扩展：地址令牌支持有效的内核对象协同管理。
记忆力	在用户空间中分页	在内核中分页	增强型：在内核中具有无策略分页的服务中进行集中式管理。
应用程序接口	POSIX兼容	POSIX+BSD/赢	扩展： Linux API/ABI通过一个兼容ABI的垫片兼容。
设备驱动程序	移植/虚拟机	本机驱动程序	增强型：通过具有双驱动程序的驱动程序容器有效地重用Linux驱动程序。



crokernels.因此，HM在强调兼容性和性能的同时，也尊重了微内核的架构设计原则。

HM在核心内核中只保留最小和必要的功能，包括线程调度程序、串行和定时器驱动程序，以及访问控制。所有其他功能都在孤立的操作系统服务中实现，例如进程/内存管理器、驱动程序和FS。此外，HM采用了细粒度的访问控制，以保留最小特权原则。服务只能访问实现功能所需的受到严格限制的资源（内核对象）。因此，HM继承了微内核的安全性、可靠性和可扩展性。

保留：最小的微内核与良好的隔离和最低特权的操作系统服务。

原则2：确定绩效的优先级。微内核的潜在好处被新出现的场景中的体系结构固有的性能问题所影响。因此，HM不是强制执行统一但过于强大的隔离，而是为组装系统提供结构支持，以满足性能和安全要求。特别是，除了采用类似rPC的快速路径来解决资源分配/耗尽/会计问题（第4.1节），HM提出了差异化的隔离类来减少IPC通过放松可信操作系统服务之间的隔离的开销（第4.2节）。HM还可以合并紧密耦合的操作系统服务（图3中的❶），以在性能要求很高的场景中最小化IPC频率（第4.3节）。此外，HM通过补充地址令牌的功能（第4.4节），实现了高效的内核对象协同管理（❷），这促进了匿名内存的无策略的内核内分页（第4.5节）。

灵活化：通过为灵活的装配提供结构性支持，以确定性能的优先级，以适应不同的场景。

原则3：最大化生态兼容性。HM通过一个重定向所有Linux的垫片（❸）来实现Linux ABI遵从性，从而与现有的软件生态系统集成
syscall到适当的操作系统服务，并作为存储和翻译Linux抽象（例如，fd），以有效地支持轮询等功能（第5.1节）。此外，HM通过驱动程序容器（❹）重用未经修改的Linux设备驱动程序，它提供了直接从主线Linux派生的必要运行时，以及少量的工程工作（第5.2节）。HM通过利用控制和数据平面分离（❺）进一步提高了驱动程序的性能。

增强：通过实现与LinuxAPI/abi兼容和性能良好的驱动程序重用，最大限度地提高兼容性。

HM的威胁模型。HM保留了微内核的架构设计原则，从而维护了与SOTA微内核类似的威胁模型，从而防止恶意应用程序和操作系统服务访问他人的内存，并确保了数据的机密性、完整性和可用性（CIA）属性，但存在以下差异。

首先，由于新兴场景中的应用程序出于兼容性的原因需要集中的内存管理(第4节。内存管理器5)（唯一的例外），包括它合并的服务（只有在部署中的手机中的FS），可以不可避免地访问应用程序的地址空间。此外，在内存是自管理的安全关键场景中，HM不会创建这样一个集中式内存管理器。

此外，为了提高性能，存在其他攻击面（4.2节）、故障域的不同分区（4.3节）和精心选择的对象上的额外数据泄漏可能性（不会破坏内核，4.4节）。详细的安全设计将在相应的章节中进行讨论。

4红孟的性能设计

.14个同步的类似RPC的IPC快速路径

微内核使用IPC来调用操作系统服务。大量的研究已经提出了许多优化，以最小化IPC开销。然而，当将它们应用于新出现的场景时，我们遇到了几个严重的问题，要么是以前被忽视的，要么是由变化的假设引起的。HM仔细地解决了这些问题，如表2所示。

表2：HM中IPC的比较。

	IPC快速路径	迁移	宏孟IPC
旁路调度	是	是	是
减少的开关	N/A	寄存器	注册。/Address Space/Priv.
资源分配	异形前	异形前	预绑定和自适应
资源耗尽	封锁的	封锁的	预留用于回收
资源会计	时间的	时间的	临时雇员/Energy/Memory

同步RPC或异步IPC。IPC通常假定具有相同执行模型的对称端点。因此，以前的工作表明，异步IPC可以避免在多核[30]上进行序列化，允许两个端点在不阻塞的情况下继续执行。然而，在新出现的场景中，我们观察到大多数ipc都是过程调用，其中可以清楚地识别调用者和被调用者。此外，操作系统服务大多是被动调用的，而不是连续工作的，而且应用程序的大多数后续操作都取决于过程调用的结果。因此，同步远程过程调用（RPC）是一种更适合用于服务调用的抽象方法。

HM采用类似rPC的线程迁移[33,94]作为服务调用的IPC快速路径。当发送IPC时，核心内核执行直接切换（绕过调度，类似于之前的工作[10、30、49、67、70]），并且只切换堆栈/指令指针（避免切换其他寄存器）以及保护域。具体来说，HM要求OS服务注册一个处理程序函数作为入口点，并准备一个执行堆栈池。当应用程序调用服务时，核心内核将调用者的堆栈/指令指针记录在调用堆栈中，并切换到具有准备好的执行堆栈的处理程序函数。在返回时，HM从调用堆栈中弹出一个条目，并切换到调用者。IPC参数主要通过寄存器传递，附加的参数通过共享内存传递。

*性能差距。*尽管HM绕过了调度并避免了交换寄存器，但由于特权级别/地址空间切换和缓存/TLB污染[9,30,49,86]（占IPC总成本的50%），它仍然面临着显著的性能下降。我们在第4.2节中使用不同的隔离类进一步弥补了这一性能差距。

资源分配。*IPC的内存占用在很大程度上被以前的工作忽略了。*然而，由于在智能手机等新兴场景中，极高的IPC频率和大量的连接（>1k线程同时出现），我们发现必须考虑IPC的内存在生产中，因为它会导致严重的问题，如内存不足（OOM）甚至系统挂起。尽管HM中的每个IPC连接只需要一个单独的执行堆栈（而不是一个包含所有相关数据结构的完整线程），但给定，它的内存占用仍然很少
大量的ipc。

以前的工作预先分配了一个固定大小的线程/堆栈池，并将其绑定到连接中。但是，由于工作负载的多样性和动态性，包括正在运行的线程的数量和对不同操作系统服务的需求，因此它的大小很难确定。大池会迅速消耗内存，而连接上的动态分配会在IPC的关键路径上引入运行时开销。我们最初尝试在创建时为每个线程中的每个操作系统服务中准备和绑定堆栈。然而，我们很快意识到这个问题仍然存在，因为有些服务几乎没有被一些线程使用（浪费），而且存在许多IPC链（到另一个操作系统服务）需要另一个堆栈。

因此，HM通过在每个线程使用的常用操作系统服务（例如，进程/内存管理器和FS）中预绑定堆栈，同时仍然维护一个堆栈池，并在运行时自适应地调整其大小，从而达到了一个最佳位置。当剩余的堆栈低于阈值时，操作系统服务将分配更多内容以减少同步分配。HM通过在IPC链中调用相同的服务（例如，类似于aba的调用）时重用相同的堆栈，从而进一步减少了它的内存占用。

资源耗尽。*IPC可能由于资源耗尽而失败。*具体来说，当OOM发生时堆栈池耗尽时，OS系统服务无法分配一个新的堆栈来处理IPC请求。然而，应用程序不能处理这样的错误（不存在于单片内核中）。因此，这样的请求在SOTA微内核中被排队（阻塞），这可能会导致严重的问题，如循环等待，甚至系统挂起。

一种直观的方法是将另一个IPC发送到内存管理器，以同步回收一些内存。然而，我们发现在这种情况下（已经是OOM了），到内存管理器的IPC很可能会再次失败。这种失败很可能发生在新兴的场景中，即工作负载是不确定性的和频繁出现的重负载（例如，同时打开多个应用程序）。

HM通过保留单个的堆栈池来减轻这种情况。一旦发生OOM，内核将使用内存回收池同步到内存管理器（重复），直到用户的IPC成功。因此，应用程序的ipc可以保证得到正确的处理。

资源会计。IPC在处理请求时假定一个不同的执行实体，从而将消耗的资源归因于操作系统服务。然而，由于新兴场景中的竞争应用程序需要对资源进行清晰的核算，因此所消耗的资源应该精确地计入调用者应用程序。以前的工作通过继承调用者的调度上下文[70,80]来实现时间隔离。然而，新出现的场景也需要计算能量和内存消耗。因此，HM记录

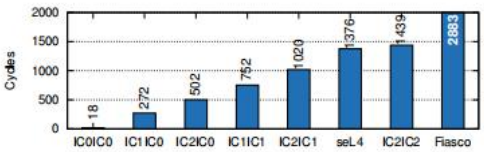


图4：树莓派4b中ICx和ICy（ICxICy）之间的往返IPC延迟。IC0包括核心内核。IC2包括用户应用程序。Zircon不能在Pi4b上运行，而且[49]要慢好几倍。

用户应用程序的身份（IPC链中的根调用者），并在处理IPC时将消耗的资源属性给它。

决定：补充async./sync。IPC具有类似RPC的快速路径，用于调用操作系统服务，同时仔细解决资源分配/耗尽/会计问题。

.24个差异化的隔离类别

OS服务的隔离。将所有操作系统服务放在用户空间中可以提高安全性，但在新出现的场景中却不能满足性能要求。*我们观察到，并非所有的服务都需要相同类别的隔离。*特别是，在实际部署中，成熟的、经过验证的和性能关键的操作系统服务可以受到较弱的隔离，以获得最佳性能。此外，快速发展的服务可能经常引入bug和漏洞，因此需要更健壮的隔离，以防止内核损坏。具有大型代码库和繁琐特性的操作系统服务，如驱动程序，需要隔离，以减少受信任的计算基础（TCB）。

因此，HM采用差异化隔离类（IC），为不同的操作系统服务提供量身定制的隔离和性能。具体来说，隔离类对服务进行分类，并定义它们之间的隔离。图4显示了与seL4[67]和Fiasco相比，在不同隔离类下的服务之间的往返IPC延迟。OC [69]。

隔离等级0：核心TCB。IC0适用于经过仔细验证的、性能极其关键的、受信任的操作系统服务，例如兼容abi的垫片（部署中唯一的IC0服务）。在这些服务和内核之间不强制执行隔离。因此，IPCs都是间接的函数调用。

IC0威胁模型： IC0是核心TCB的一部分，任何受损的IC0服务都可以任意读取和修改他人的内存。因此，在IC0上放置服务应该经过仔细的验证，以避免核心内核损坏。

隔离类别1：机制强制执行的隔离。IC1适用于性能关键型和经过验证的操作系统服务。受以前的内核内隔离方法[11,49,59,71,112,120]的启发，HM将这些服务放在内核空间中，并使用机制来强制执行服务之间的隔离。具体来说，HM仔细地将内核地址空间划分为不同的域，并为每个服务分配一个唯一的域（IC0/核心内核也驻留在一个唯一的域中）。HM使用ARM监视点[63]和Intel PKS [60]来防止跨域内存访问。此外，由于IC1服务在内核空间中运行，因此它们可以执行特权指令。为了防止这种情况发生，HM采用了二进制扫描和轻量级扫描控制流完整性（CFI，利用ARM指针身份验证（PA）[77]）来确保服务不能执行包含特权指令的非法控制流，并使用安全监视器[49,108]来保护页表，防止代码注入，这也通过VM退出捕获任何特权指令，作为对CFI的补充。

IC1服务（或到IC0）之间的IPC将进入核心内核中的一个门，该门执行最小的上下文交换（交换指令和堆栈指针，没有地址空间交换和调度），并将硬件配置为交换域（只需要几个周期）。不能绕过这样的门，因为域交换机需要特权指令。因此，IPC开销显著降低。如图4所示，与用户空间服务（IC2IC2）相比，它将IC1服务之间的IPC延迟减少了50%。

IC1威胁模型：IC1的威胁模型不同于其他模型通过假设所应用的隔离机制的正确性、可靠性和安全性，这确实暴露了一些额外的攻击面。例如，最近出现了针对ARM PA的新攻击。除此之外，IC1共享相同的威胁模型，它禁止任何受损的服务读取/写核心内核的内存（和其他操作系统服务的）和执行特权指令。

隔离类别2：地址空间隔离。IC2适用于非性能关键型服务或那些包含第三方代码（例如，Linux驱动程序）的服务，由地址空间和特权隔离强制执行。HM（IC2IC2）中IC2服务之间的IPC比seL4中略慢，这主要是由于细粒度锁定，这对于在实际负载下扩展到多核进程是必不可少的。

IC2威胁模型： IC2与其他多服务器微内核共享完全相同的威胁模型。

虽然IC1显著降低了IPC开销，但它也引入了额外的攻击面并有资源限制（例如，Intel PKS中的16个域，ARM监视点中的4个域）。因此，只有性能关键的和经过验证的操作系统服务被放置在IC1上。此外，如果出现新的攻击，HM可以快速地将所有服务移动回IC2。我们将在第4.3节中进一步讨论关于配置隔离类的部署经验。此外，IC0/1并不意味着耦合到内核。隔离类允许在部署期间进行可配置的隔离决策，而不是在开发期间进行隔离假设。不同的场景使用不同的配置，如图3所示。

决策：并非所有的操作系统服务都需要相同类别的隔离。采用差异化的隔离类来放松受信任服务之间的隔离，以提高性能。

.34灵活的组成

操作系统服务的分区。虽然直观地说，操作系统服务应该很好地解耦，例如，FS和内存管理器，我们观察到操作系统服务是不对称的，因为一些

功能需要特定服务之间的密切合作。例如，FS并不是访问文件的唯一入口。POSIX支持通过内存管理器读取文件的文件映射，并且它经常出现在关键路径上，并显著影响用户体验。

隔离类之间强制执行相同的隔离
相同类别的操作系统服务。因此，如果没有进一步的结构支持，HM与单片内核相比仍然面临着性能下降。首先，即使在IC1（内核空间）中，紧密耦合服务之间经常调用的ipc仍然会造成明显的开销（内存映射文件的页面故障处理率为20%）。此外，对共享状态的双重簿记，如页面缓存，还引入了大量的内存占用和同步开销。最后，没有页面缓存的全局视图来指导资源回收（例如，最近使用的，LRU）。

为了弥合性能差距，HM采用了一种可配置的方法，允许在性能要求很高的场景中合并紧密耦合的操作系统服务，权衡隔离以获得更好的性能，同时保留在安全关键场景中分离它们的能力。当合并时，不强制隔离，两个服务之间的ipc变成函数调用，而其他服务则保持不变（完全隔离）。

合并还可以实现对页面缓存的高效协同管理。在合并FS和内存管理器时，它们可以同时共同管理它们，而不是在FS和内存管理器中同时共同管理它们。它消除了双重簿记和同步开销，并为有效的回收提供了一个全局视图。为了保持分离它们的能力，我们提供了一种机制，在分离时自动将共享页面缓存的访问转换为ipc。然而，它将引入一些重要的开销。因此，在部署过程中，我们都实现了这两个版本。/shr.)手动操作，并相应地启用它们。

表演如表3所示，当将FS与内存管理器合并时，替换IPC将由于页面缓存失败而导致的页面故障的处理延迟减少了20%(9月。*藏物处*它可以进一步降低30% (Shr。缓存)，并通过共同管理共享的页面缓存，实现了与Linux（5.10，详见第6.2节）类似的性能。合并还可以将tmpfs的写吞吐量提高40%。此外，在智能手机中，合并服务的内存占用减少了37%（FS+内存）。

保护措施合并后的服务位于一个故障域中，其威胁模型（作为一个整体）与它所在的隔离类保持相同。因此，任何失败或损坏的服务都只能破坏其合并的服务，这也是性能的主要妥协。因此，应该仔细评估服务合并。在实践中，由于智能手机中文件操作的频率极高（图1e），它们的性能目标只能通过将FS与内存管理器合并来实现。然而，与单片内核相比，安全性仍然得到了改进（与其他服务隔离）。

部署经验。与分化的

表3：通过在麒麟9000[57]的大核心中合并FS服务和内存管理器，提高了性能。

分离合并的Linux			
页面故障（周期）	7092	5290 (Sep.缓存)3785(Shr。藏物处	3432
Tmpfs写入（MB/s）内存足迹（MB）	1492 190	2067 120	2133 N/A

表4：地址令牌支持大多数功能的操作，并允许直接访问，除了限制细粒度的操作和链撤销。

	能力	地址令牌
词元	CSlot id	映射地址
访问权限	委托到内核	直接（RW）/写（RO）
所有权	CNode中的上限	映射的页面
拨款	移动到CNode	将页面映射到VSpace
撤消	从CNode中删除	取消映射页面
链撤消	支持	不支持
保护措施	监控所有操作	映射Obj的限制。

隔离类，HM支持灵活的组合，允许关键组件被灵活地组装（用户空间或内核空间，分离或合并），支持根据场景的需求探索隔离和性能之间的权衡，并能够从路由器扩展到智能手机与相同的代码库。HM的演变见证了这样的探索。最初，所有的服务都在IC2处被隔离了。为了达到性能目标，我们仔细地组装了系统，通过保留以下规则来保留大部分的安全属性。

首先，由于附加的攻击面，IC1服务不能包含任何第三方代码。因此，尽管一些驱动程序也是性能关键的，但我们将它们保持在IC2，并通过控制/数据平面分离加快速度（第5.2节）。其次，服务的合并，特别是与内存管理器的合并，不可否认地削弱了隔离性和安全性（尽管与单片内核相比仍然有所改进）。因此，我们让它保持可配置状态，并且只在手机上启用它。此外，ICO不仅增加了核心TCB，而且有严格的内存限制和非阻塞要求。因此，在实践中，HM只将ABI垫片（可以选择退出）放置在ICO中。第6.1节详细介绍了这些配置。

决策：操作系统服务是不对称的。协调紧密耦合的操作系统服务，并灵活地组装系统，以满足各种场景下的不同需求。

.44基于地址令牌的访问控制

SOTA微内核使所有内核对象显式，并服从基于功能的访问控制[30]，以保持最小特权原则，这主要以分区方式实现，在表示访问内核对象权限的用户空间中保留一个令牌（通常是插槽ID）。然而，当我们在新兴的场景中部署它时，我们遇到了严重的性能问题。

关系清晰，但访问速度缓慢。*尽管功能在描述内核对象的外部关系方面很有效，即授权链，访问它们的内部关系*

内容需要将带有操作的令牌发送到核心内核，由于特权切换和访问多个元数据表，这带来了重要的性能开销。内核对象隐藏在这些功能后面，只有核心内核才能访问。然而，由于最小性原则，一些内核对象的内容(e。g.，页表)应该经常被核心内核之外的操作系统服务更新，因为分区功能不再有效。

一些微内核通过将特定的对象映射到用户空间来加快访问速度。但是，为了安全，它们只能应用于少数有限的对象（例如内存对象[30,67]、线程控制块[3,9,76]的部分和内核接口页面[76]），并且缺乏正确同步数据的能力，这抑制了内核和操作系统服务之间的合作。为了解决这些问题，HM提出了一种基于地址令牌的通用方法，该方法可以应用于更广泛的对象，从而实现有效的协同管理。

具体来说，如图5所示，每个内核对象都被放置在HM中的一个唯一的物理页面上。将内核对象授予操作系统服务需要将这样的页面映射到其地址空间（❶）。因此，映射的地址作为令牌，直接从硬件访问内核对象，而不涉及内核（不情愿地）。可以授予内核对象（映射）为只读（RO）或读写（RW）。操作系统服务可以读取RO内核对象，而无需内核参与。要更新它们，应该使用一个新的系统共享，writev，用更新的值传递目标地址，核心内核将通过引用内核对象的元数据（❷）来验证权限。对于RW内核对象，一旦被授予，可以通过操作系统服务更新而无需内核参与（❸）。此外，对于小于具有相同属性（权限）和类似生命周期的页面的对象，HM在分配时将这些对象批处理到单个页面中，允许它们被集体授予和撤销。

功能。地址令牌支持大多数功能的操作，如表4所示（与seL4 [107]相比，Zircon具有类似的功能[37]），但有两个例外。首先，地址标记一旦被授予就不能限制细粒度操作，这将削弱安全性并暴露额外的攻击面。此外，功能存储详细的关系，允许链撤销，由于隐式所有权，不支持地址令牌。然而，地址标记只被选定的共同管理的内核对象使用。小心地减轻攻击面（下面讨论）。
此外，由于集中的资源管理，内核对象具有特定的所有者（不会授予其他对象）。因此，很少使用链撤销。

保护措施一旦将地址令牌授予操作系统服务，内核就无法监视后续操作。HM通过限制映射到用户空间的对象（通过静态分析强制执行）来缓解这种情况。只有在内核保留的结构中只包含某些变量的值的内核对象(不允许指针阻止时间到-

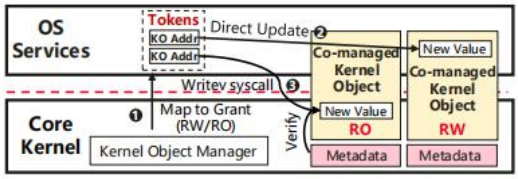


图5：HM中基于地址令牌的访问控制。❶映射内核对象的页面要授予。❷直接访问RW对象。❸使用

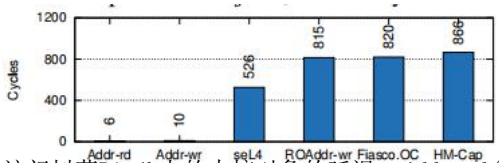


图6：访问树莓Pi 4b上的内核对象的延迟。*Addr-rd/wr表示HM中的地址标记*。ROAddr-wr表示对HM中的只读对象的写入。

检查使用时间攻击)映射RW(例如，第4节中的PCache。5)，确保它们不会使用不正确或不一致的数据损坏内核。内核的其他内部状态（例如，指针和引用计数器）只能被映射为RO或根本不被授予，以防止它被损坏。当读取RW对象时，HM进一步应用了完整性检查。它确实通过泄漏内核内部信息引入了一些攻击面，这可以通过ARM PA等硬件加密来减轻。

同步。共享数据有两种方法
在操作系统服务和内核之间利用地址令牌。首先，操作系统服务和内核可以异步地交换消息（消息传递）。例如，第4.5节中的PCache将预先分配的页面发送给内核，以便将来进行内核分页。HM使用一个无锁的环缓冲区来正确地同步数据。此外，OS服务还可以对内核可以并发读取的对象（例如，第4.5节中的VSpace，它存储内存布局）应用本地更新。HM采用细粒度锁定，以确保其正确性。但是，当服务在执行关键部分时已被抢占时，它可能会阻止内核。因此，内核只能在rw映射的对象上使用trylock操作。如果失败，HM将重定向到操作系统服务（慢路径）以完成该过程（例如，第4.5节中的分页）。

表演图6比较了应用地址令牌后访问内核对象的延迟。与基于能力的方法相比，阅读和写作（到RW）的延迟显著减少。然而，在RPi4b上写入RO对象的延迟比seL4慢，主要是由于在ARM上使用了AT指令来转换地址和检查权限，这在RPi4b上很慢（但在高级智能手机芯片中进行了优化）。

使用情况。对于安全问题（见上文），地址令牌是操作系统的内部抽象，它补充了与操作系统服务进行高效协同管理的功能。具体来说，除了启用对由服务管理的内核对象的直接更新之外，它还允许它们读取内部状态(e。g.，没有内核参与，sim-

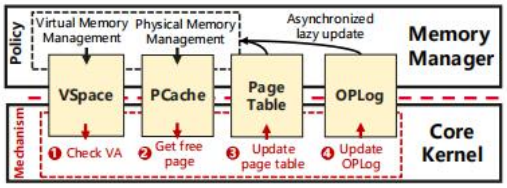


图7：HM中的无策略内核分页。在页面故障时，内核检查地址❶，如果是匿名的，则❷/❸映射一个预先分配的页面，并且❹记录一个OPLog。

加载到Linux [82]中的虚拟动态共享对象（vDSO）。它还允许完全在内核中处理性能关键事件（例如，第4节中的页面故障），而不违反最小化原则，即通过（通过服务）提前做出策略驱动的决策，并通过共同管理的对象与核心内核进行通信。.5

决策：隐藏内核对象的功能会将内核（不情愿地）插入到数据平面上。补充以地址令牌，以实现有效的共同管理。

.54无政策内核分页

集中管理vs。分布式寻呼机。一些SOTA微内核（例如，seL4）将内存管理委托给具有单独自定义寻呼机的应用程序。然而，由于新兴场景中的竞争应用程序需要协调和集中的管理，我们发现很难有效地实现某些需要内存的特性，这些特性需要分散寻呼器的内存全局视图，例如控制组（cgroup）和内存循环。因此，HM通过一个集中式的内存管理器来管理内存。为了实现最小化，内存管理器在核心内核之外，核心内核管理物理内存和虚拟内存，并处理所有应用程序和操作系统服务的页面故障。

缓慢的用户空间分页。我们观察到，由于匿名内存（例如堆栈/堆）的分页过程缓慢，应用程序内存)的性能显著下降，这在智能手机中经常发生，如图1b所示。退化主要是由于从内核到寻呼机的额外往返旅行。具体来说，在抛出页面错误异常之后，内核向寻呼机发出一个IPC，寻呼机检查地址并分配一个新页面，然后返回到内核更新页面表，最后返回到应用程序。这样的往返是不可避免的，因为页面故障处理涉及一个策略决定是否和物理页面映射，应该远离内核[27]，而异常处理内核，和页面表隐藏在内核的功能。

为了提高匿名内存页面故障的处理性能，HM进行策略驱动决策
提前，并在核心内核中留下了一个无策略的页面故障处理机制。因此，它消除了在关键路径上额外的IPC往返。具体来说，内存管理器提供了匿名内存的地址范围

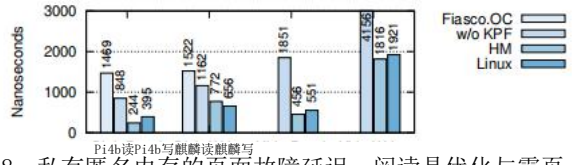


图8：私有匿名内存的页面故障延迟。阅读是优化与零页。seL4不包括在内，因为默认情况下不支持需求分页。

有几个预先分配的物理页面。如图7所示，如果页面故障在范围内（❶）被触发，核心内核可以直接将其映射到预先分配的物理页面（❷和❸），并记录操作日志（OPLog，❹），内存管理器将使用它异步更新其内部状态（例如，映射的匿名页面的计数器）。否则，如果地址超出指定范围（非性能关键）或者预先分配的页面耗尽，核心内核将向内存管理器进行IPC。所涉及的内核对象由内存管理器通过地址令牌共同管理，包括页表、操作日志、记录用于识别匿名内存的虚拟内存空间布局的VSpace，以及存储预分配的页面的PCache。

妥协。通过提前做出策略驱动的决策，策略（是否//哪个映射）仍然保持在核心内核之外。唯一受损的能力是在预先分配给PCache后更改策略，这降低了灵活性。PCache还引入了一些额外的内存占用。然而，由于PCache可以定期补充（脱离关键路径），它的大小仍然相对较小，这使得这些权衡可以接受。

表演图8显示了HM中内核分页（KPF）延迟的减少。HM在Pi4b上减少了72%/33%，在麒麟9000（小核心）上减少了75%/55%，使其甚至比Linux略短(6。1在Pi4b和5上。10在麒麟9000)。不包括seL4，因为它需要自定义寻呼机，并且默认情况下不支持需求分页。在Pi4b上，故障处理的往返（到寻呼机）上大约需要140ns（使用sel4工作台测量），这使得它比Linux慢得多。

决策：通过抢占策略驱动的决策，启用无策略的内核分页。

5红孟的兼容性设计

.15 Linux ABI兼容性

在新兴场景中部署需要Linux ABI兼容性，这在多服务器微内核中带来了挑战。K42 [68]通过陷阱反射实现了Linux ABI兼容性，它将系统重定向回加载到应用程序地址空间中的K42库。然而，由于到内核[2]的额外往返，它引入了显著的性能开销，而且也面临着实现-

我挑战[29,111]，因为在用户空间中保持状态。FreeBSD [36]和Windows（WSL1 [87]）也通过系统扫描仿真实现了部分Linux ABI兼容性。然而，由于它们所有的操作系统服务都位于内核空间中，仿真层可以将fd等抽象直接映射到其内部状态，并有效地支持fork和poll等功能，这在多服务器微内核中具有挑战性。

Syscall重定向。HM通过在ICO（内核空间）中放置一个符合ABI的垫片来实现Linux ABI兼容性，它将Linux系统重定向到适当的操作系统服务（通过系统扫描号标识，本机系统尺度绕过垫片），如图3a所示。此外，垫片是可选的。在应用程序主要是自定义的场景中，HM用兼容posix的库替换了垫片，如图3b所示。

集中的状态。除了二进制兼容性之外，微内核不再有一个用于全局状态的中心存储库，例如文件描述符（fd）表，使像fd复用(i。e., 轮询)和像fork这样的系统尺度[2]实施具有挑战性。具体来说，fd表通常保存在应用程序的地址空间中（只包含由操作系统服务验证的凭据）。因此，fd多路复用需要将所有等待的fd映射到一个通知原语，并将其发送到所有相关的服务。此外，像fork这样的系统尺度必须在用户空间中正确地组装这种分布式状态。它引入了显著的复杂性和性能开销，主要是由于将状态从父状态传递到子状态，以及由于更新这些写时复制状态[8,29]而导致的额外页面错误。因此，SOTA微内核，包括seL4、Fiasco和Zircon，不支持分叉，而K42中的分叉已知有严重的性能问题[29,111]。

因此，HM中兼容abi的shim还可以作为fd表等全局状态的中央存储库，实现轮询等多路复用和fork这样的系统转换的有效实现。具体地说，shim维护fd表，它将fd映射到凭据（由OS服务用于标识用户）。因此，实现轮询只需要在shim中维护一个轮询列表，并通过地址令牌与OS服务共同管理。它还避免了在执行fork时在用户空间中复制fd表。

Vecyedssalls。尽管大多数系统转换转换都是在兼容abi的垫片中实现的，但也有矢量系统转换[116]（例如ioctl/fcntl）扩展系统api，并允许通过文件抽象自定义扩展（用于驱动程序/模块）。HM在FS服务中重定向和处理它们（例如，在第5.2节中调用驱动程序容器）。

部署经验。HM通过了AOSP兼容性和供应商测试套件（CTS/VTS [43,44]）中的所有测试，它检查了内核功能和驱动程序行为。尽管大多数二进制文件可以开箱可用，但我们观察到一些应用程序依赖于不稳定/无文档记录的Linux行为，并且无法在HM上运行。例如，应用程序这取决于特定的epoll返回顺序，[95]无法在HM上运行（它在不同的Linux版本中也会失败）。

决策：通过与ABI兼容的垫片实现Linux二进制兼容性。

.25驱动容器

Linux拥有最丰富的设备驱动生态系统。此外，一些驱动程序没有源代码可用，这使得移植具有挑战性。因此，重复使用Linux驱动程序对于广泛的部署至关重要。

挑战实际的和性能强的驱动程序重用。之前的工作，包括移植[3,17,32,41,118]和基于vm的方法[72]，在同时实现高兼容性、合理的工程努力和不妥协的性能方面面临着挑战。特别是，移植运行时环境需要重新实现驱动程序使用的所有内核api（KAPi）。因为一些司机使用大量的kapi，其中一些甚至在不断发展，这种方法面临着兼容性和负担得起的工程工作的挑战。此外，重用的驱动程序（具有较大的不可信的代码库）应该强制执行严格的地址空间隔离，以获得更好的安全性，并避免许可证污染[34]，这也会降低性能。通过虚拟机重用驱动程序可以用更少的人力资源来实现更好的兼容性。然而，它引入了一些问题，包括内存双重管理导致额外的内存占用（在智能手机等内存受限的场景中至关重要），以及由于驱动程序中频繁使用异步通知而降低性能。

HM通过一个驱动程序容器重用Linux驱动程序（图9），从而在兼容性、工程工作和关键路径性能之间找到一个最佳位置。

兼容性。受LKL [101]、UML [26]和SawMill [39]的启发，Linux驱动程序容器（LDC）通过重用Linux代码库作为用户空间运行时，提供了所有必要的Linux[39]，允许现有的Linux驱动程序不经修改地运行。主要区别在于LKL/UML/SawMill是指LDC重用驱动程序，而不是诸如文件系统和网络堆栈等组件。因此，驱动程序应该能够直接访问硬件设备，而不是重定向到主机驱动程序。此外，运行时还依赖于HM来进行资源管理。因此，所有相关的功能，如线程调度器，都将被删除。

HM创建另一个设备管理器，它管理Linux和本地驱动程序容器（本地驱动程序驻留的地方）。除了初始化驱动程序容器外，它还在虚拟文件系统（VFS）中注册条目（图9中的❶），以便驱动程序通过VFS (e。g., ioctl❷)可以正确地重定向到适当的驱动程序容器（❸）。

使用LDC，HM已经成功重用了来自Linux的700个设备驱动程序，包括所有需要的

[2]虽然有人认为fork应该弃用[8]，但像AOSP/Open和谐这样的流行框架仍然使用fork。

智能手机和车辆的正常工作，如摄像头、显示器、音频、NPU/GPU和存储器。虽然大多数驱动程序可以直接运行，但存在几个例外。由于LDC是在用户空间（IC2）中运行的，因此使用特权指令（例如，smc）的驱动程序将触发故障。它们需要在核心内核中进行二进制重写或手动移植（对于那些经常使用特权指令的用户，例如，GIC）。

工程工作。LDC中的Linux运行时是直接从主线Linux派生出来，稍作修改，以便将多个功能重定向到驱动程序容器库（图9中的DC-base），以便正确执行。因此，所需的工程工作规模很小。具体来说，我们提供了一个虚拟架构，并将kthead/内存接口重定向到HM。为了使驱动程序正常工作，DCbase创建了一个虚拟计时器和一个虚拟IRQ芯片来提供中断请求，并为像virt_to_phys这样的函数保留一个线性映射空间。与引入双内存管理和双线程调度的基于vm的方法相比，驱动程序容器通过在HM中重定向和管理它们，避免了这些问题。

在实践中，支持长期支持（LTS）内核发行版就足以重用大多数驱动程序(目前，HM支持4.4,4。19和5。10).此外，由于与dc基相关联的Linux接口相对稳定，因此只需要进行很小的修改就可以升级Linux运行时。从4.19升级到5.10需要少于100次的d基更改，其中大部分是对过程名称、参数和结构的微小修改。

关键路径性能。LDC被放置在IC2（用户空间）中，以保护安全性（驱动程序有大量的不受信任的代码库），并避免许可证污染。然而，它在驱动程序关键场景中引入了非常重要的开销，比如应用程序启动和摄像头。因此，HM通过在原生驱动程序容器中创建一个双驱动程序来应用控制平面和数据平面分离，该驱动程序处理性能关键路径上的I/Oirq（图9中的❹）。双驱动程序重写数据处理过程，因此可以用较弱的隔离（放置在内核空间中的IC1上）来强制执行，从而产生显著更好的性能。控制平面包含笨重的程序，如启动/暂停/恢复，保留在LDC（❺）中。

双驱动程序通过IPC同步状态（通常是一个变量）。由于控制平面完全在LDC中处理，因此双驱动程序不会修改状态（I/O错误被重定向到LDC）。在初始化时，LDC将设备信息传递给本机信息，以创建双驱动程序。当处理非I/OIRQs和错误时，LDC会将更新后的状态同步回双驱动程序。与仅在LDC中进行的透明集成不同，这会导致性能不佳，而双驱动程序需要额外的工程努力来分割和重定向中断和同步状态。因此，双驱动程序仅用于性能驱动程序，如通用闪存（UFS）驱动程序（其他驱动程序透明地集成）。

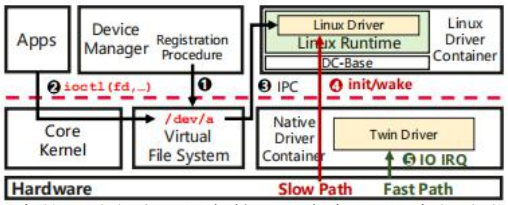


图9：HM中的驱动程序。设备管理器会在VFS❶中创建文件节点。VFS将调用❷重定向到驱动程序❸。HM通过分离控制❹/data❺平面来提高性能。

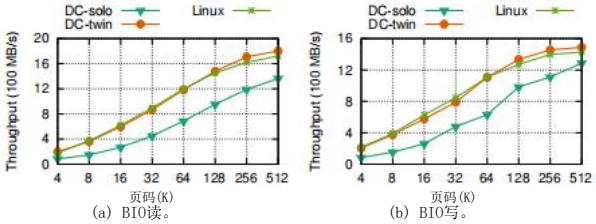


图10：麒麟9000上的块I/O吞吐量。DC-twin应用数据和控制平面分离，而DC-solo则没有。

图10显示了在UFS块I/O基准测试中改进的吞吐量。在实验中，I/O请求直接从驱动程序发出。DC-twin（应用数据/控制平面分离）实现了与Linux 5.10类似的吞吐量，并且在4K大小时比DC-solo（w/o分离）高出140%。

保护措施LDC几乎是HM中一个正常的用户空间（IC2）操作系统服务，还可以创建一个线性映射空间，该空间的范围严格限制在其分配的内存中（仅通过在分配的页面上设置当前位）。因此，它与其他微内核中的IC2 OS服务和用户空间驱动程序共享相同的威胁模型。此外，HM使用SMMU[5]来防止DMA攻击，其驱动程序驻留在一个孤立的本地驱动程序容器中。

决策：通过具有控制/数据平面分离的驱动程序容器有效地重用Linux设备驱动程序。

6野外梦

.16、实施和部署

HM的核心内核主要在C的一个限制子集中实现，由90k行代码（LoC）组成，其中包括基本功能。所有其他的操作系统服务都是解耦的，可以单独部署，总计超过100万的LoC。HM的构建系统可以根据针对各种场景指定的详细配置来组装操作系统服务，例如将操作系统服务置于不同的隔离环境中
类或合并一些操作系统服务。

HM已经部署在各种新兴场景中的数千万台设备中，这些场景共享相同的代码库，但具有不同的配置。在安全关键场景中，如智能车辆（仪表盘和娱乐系统）和智能手机的可信执行环境（TEE），安全和严格的隔离优先于性能。此外，应用程序大多是自定义的

表5：LMbace结果。

基准命令[1]	单元	一种可免费使用的操作系统	*hm*	范数[2]
lat_unix -P 1	μs	10.23	10.39	0.98
lat_tcp -m 16	μs	21.22	17.19	1.23
lat_tcp -m 16K	μs	24.54	18.9	1.29
lattcp-mlK（同芯）	μs	21.21	17.19	1.23
lat_tcp-mlK（跨铁芯）	μs	37.96	25.66	1.47
lat_udp -m 16	μs	17.83	19.48	0.92
lat_udp -m 16K	μs	23.63	22.02	1.07
lat_udp-mlK（同芯）	μs	18.04	19.55	0.92
latudp-mlK（跨芯）	μs	34.17	26.84	1.27
bw_tcp -m 10M	MB/s	1812	3109	1.71
bw_unix	MB/s	7124	8478	1.19
bw_mem 256m bcopy	MB/s	17696	17202	1.02
bw_mem 512m frd	MB/s	14514	14593	0.99
bw mem 256m fcp	MB/s	17492	15867	0.91
bw_mem 512m fwr	MB/s	34771	35318	1.01
bw_file_rd 512M io_only	MB/s	8976	9396	1.04
bw_mmap_rd 512M mmap_only	MB/s	26073	27520	1.05
lat_mmap 512m	μs	3315	3628	0.91
lat_pagefault	μs	0.83	0.78	1.06
lat_ctx -s 16 8	μs	4.53	3.41	1.32
bw管	MB/s	3808	4127	1.08
lat_pipe	μs	9.00	7.88	1.14
lat_proc执行	μs	336	1305	0.26
拉特普罗克叉	μs	323	1280	0 25
lat_proc外壳	μs	2269	4778	0.47
lat_clone（创建线程）	μs	28.6	54.3	0.52

[1]此时，将省略了参数“-P1”。

[2]*范数*显示了规范化的性能。对于吞吐量，请使用HM/Linux，对于延迟，请使用Linux/HM。越好。

和源代码。因此，HM将所有操作系统服务放在IC2（用户空间）中，并通过库将POSIX API公开给应用程序。此外，HM通过在TEE中引入驱动器微重启来实现容错能力。TEE中的驱动程序可以被认为是无状态的，因为只需要重新初始化来恢复已损坏的驱动程序。通过微重新启动，TEE可以在几百毫秒内从驱动程序损坏中恢复，而对于单片内核则需要一个完整的系统重新启动。对更广泛的场景的容错能力(e。g.，富操作系统中的有状态操作系统服务)需要额外的努力来存储状态并保持它们的一致性，我们将这些工作留给未来的工作。

在智能手机等性能要求很高的场景中，HM将性能关键的操作系统服务放在IC1（内核空间）中，包括进程管理器、内存管理器、FS和本机驱动程序容器，并将FS与内存管理器合并。Linux驱动程序容器和其他非性能关键的操作系统服务，如CPU频率调节器和电源管理器，仍然保持在IC2（用户空间）中。

.26性能

我们展示了HM和Linux在新兴场景中的端到端性能比较，包括智能手机（使用麒麟9000SoC[57]）、智能汽车和智能路由器，这是现有的微内核无法支持的。比较Linux 5。10个对应版本已经高度优化（在以前的产品中使用），而不是香草版本。

LMbace。我们在麒麟9000上使用LMbench [85]来评估了基本的操作系统功能。表5显示了与操作系统体系结构相关的结果。与Linux (5。10)，上下文交换lat_ctx（32%）和网络(平均。21%)在HM上速度更快，主要是由于与Linux [89,96]相比，它简化了处理过程。内存操作执行类似于Linux。尽管fork在微基准测试中的性能仍然比Linux差，但我们观察到，在实际加载过程中，fork的主要开销来自于复制虚拟内存区域（vma）。它可以通过并行性来加速，这将其开销从150 ms减少到60 ms（在典型的应用程序中，在Linux中接近30 ms）。克隆（创建线程）也比Linux慢1倍，这主要是由于在多个操作系统服务（特别是IC2中的驱动程序容器）和核心内核之间有额外的ipc。

极客席。图11c显示了cpu密集型的极客板凳5.3.2 [99]的标准化单孔结果。通过对系统进行性能优先级组装，HM实现了与Linux相似的性能，但由于不同的CPU频率改变策略，性能略有不同。

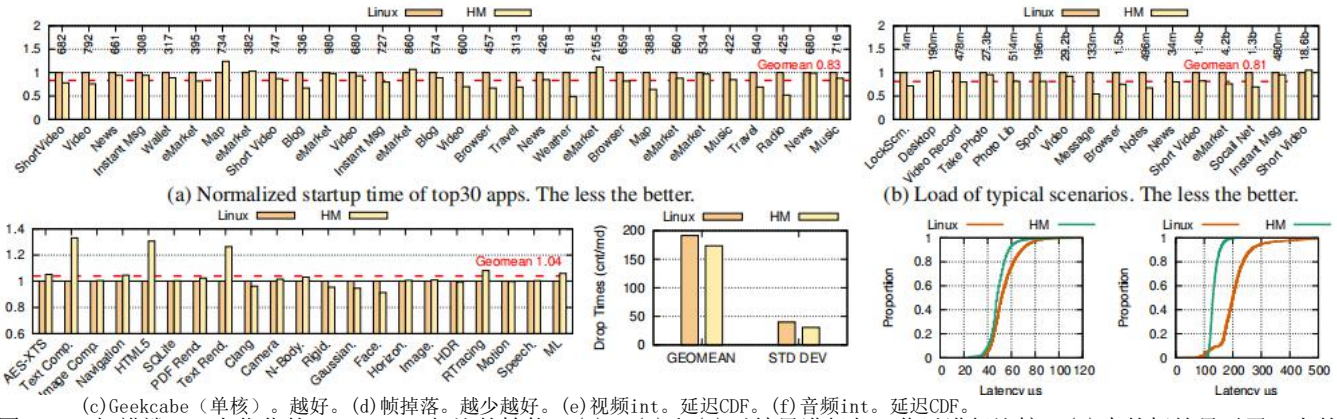
应用程序冷启动时间。应用程序的启动时间对用户体验至关重要，它强调了多个操作系统服务(e。g.，从闪存中读取和创建线程)与广泛的ipc。图11a显示了HM上前30个AOSP应用程序的启动时间。框架/应用程序版本在Linux和HM上是相同的。正如在第3.1节中所分析的，在这种情况下，微内核的主要开销来自于状态的双重簿记和缓慢的分页，而HM则消除了这一点。因此，启动时间甚至比Linux短17%（几何平均值），这主要是由于负载较轻（见下文）和自定义调度策略。

应用程序加载。图11b显示了在不同场景中的一个周期内的负载。加载（执行指令的数量）是使用perf收集的，其中包括操作系统服务（或Linux内核）中执行的指令。HM上的负载比Linux上的轻19%（几何平均值）。在HM中提出的技术显著降低了最小化和细粒度访问控制的开销。更轻的负载也使HM能够实现比Linux更好的性能和能源效率。

我们进一步在HM中使用自定义策略进行改进，这在Linux中应用具有挑战性（第2.3节）。 **帧掉落。**图11d显示了在运行第3.1节中的典型使用模型的20轮过程中的帧丢失时间（对用户体验至关重要）。*由于较轻的负载和自定义的qos引导的调度，帧下降10%*比Linux少，稳定20%。

中断延迟。图11e和11f显示了在播放视频和音频时，相关中断的延迟CDF，这对用户体验至关重要。*HM通过使用自定义体验优先策略，减少了10%（视频）和65%（音频），一次执行所有处理过程，这在Linux的另一个附加中断（由于延迟禁用ARM GIC [40]）中处理。*

在智能路由器和智能汽车方面的经验。在智能路由器中，HM减少了30%的操作系统内存占用，允许增加30%的客户端连接。在智能汽车中，HM将系统冷启动时间从1.5秒（Linux）减少到0.6秒（对用户体验至关重要，例如，启用360度环绕视图），并减少了跨域(仪表盘和娱乐-



(a) Normalized startup time of top30 apps. The less the better.

(b) Load of typical scenarios. The less the better.

(c)Geekcabe（单核）。越好。(d)帧掉落。越少越好。(e)视频int。延迟CDF。(f)音频int。延迟CDF。

图11：HM与麒麟9000上优化的Linux 5.10相比的性能。(a)、(b)和(c)对结果进行归一化以进行比较。(a)中的标签显示了HM上的启动时间，单位为毫秒。(b)中的标签显示了在HM上执行的指令。

从250µs到100µs的通信延迟。

7教训和经验

先兼容，然后逐渐适应。兼容性是商业部署的一个关键的第一步。首先，为了提高成本效益，产品通常更喜欢针对各种平台的统一代码基础。其次，一些第三方应用程序/驱动程序以二进制文件的形式分发。此外，即使目标是为了重建一个新的软件生态系统，许多必要的库仍然需要Linux兼容性。因此，只有首先保持兼容，一个新的操作系统才能被广泛部署，并有机会转向本机接口以提高性能。

单是规格说明是不够的。通过大规模的测试来检查兼容性。实现完全的兼容性是困难的（如果可能的话），主要是由于Hyrum定律[121]，它揭示了所有可观察到的系统行为都将依赖于此。因此，我们不满足某些规格，而是满足通过大规模的测试来检查兼容性，这对于发现隐藏的兼容性问题是必要的。

先进行部署，然后进行持续进行优化。微内核最初很难满足所有的性能目标，需要全系统优化(e。g.，框架，甚至硬件)。如果不进行部署，则促进多个团队之间的合作以进行这种优化是困难的。此外，生产部署还需要花时间来测试可靠性。因此，部署应尽早开始，从小规模开始。**尽可能多地使用自动验证。**我们发现，由于代码大小和功能的快速增长，完整的形式验证（使用交互式定理证明）是不可持续的。因此，我们采用了关键组件的半正式验证，并使用自动化验证和验证导向的测试来提高代码质量。

由于规模效应而导致的硬件故障/错误的放大。我们发现，一些低概率的硬件故障或错误在大规模部署时相对有可能发生，显著影响用户体验，并有可能成为在安全关键的情况下会致命。HM通过隔离不同最不发达国家中的关键驱动程序，重新启动TEE中的无状态驱动程序，以及创建用于监控的监督机构来缓解这些问题。HM作为一个微内核，在未来的工作中通过架构设计来解决这些问题的机会。**大的内核锁在新兴的场景中是不可扩展的。**虽然有人认为一个大的内核锁对于微内核[97]具有足够的可伸缩性，这主要是由于大多数系统扩展的持续时间较短，但我们发现它在手机上仍然面临着可伸缩性问题。*首先，手机有一个很高的系统通频频率（61k/s，图1f），这引起了显著的争论。此外，新出现的场景需要一些具有长时间的复杂功能。*例子包括轮询，它需要同步垫片（ICO）中的大量状态，以及能量感知调度[115]，由于手机上线程的短期运行特性，它涉及到每个调度决策的频繁和昂贵的功耗计算。

8.结论和未来的工作

宏孟是一个商业化的通用微内核，它保留了微内核原则，同时提供了结构性支持，以解决新兴场景中的兼容性和性能挑战。它也有助于未来探索微籽粒在生产中的好处。例如，它的灵活性提供了机会，以适应Linux无法解决[104]的日益增长的硬件异构性的问题，并实现容错以提高可用性。

致谢

我们感谢我们的牧羊人蒂莫西·罗斯科和匿名审稿人的深刻评论。我们也感谢华为OS内核实验室和上海交通大学的同事们提供的有益工作和支持。陈海波还获国家自然科学基金资助。61925206和62132014)。

参考文献

[1]实时Linux。https://wiki .linuxfoundation .org/realtime/start.2024年4月16日通过。

[2] J.Appavoo, M.Auslander, M.Butrico, D.M.达席尔瓦，O.克里格，M。F.Mergen。奥斯特罗夫斯基，B。罗森堡R.W.怀斯涅夫斯基和J。西尼迪斯。K42是一个开源的、兼容linux的、可扩展的操作系统内核。*IBM系统。J.*, 44(2):427 - 440, jan 2005.

[3]，乔纳森·阿帕沃，马克·奥斯兰德，迪尔玛·达席尔瓦，
大卫·埃德尔森、奥兰·克里格、米查尔·奥斯特罗斯基、布莱恩·罗森伯格、威斯涅夫斯基和吉米·希尼迪斯。利用K42中的Linux内核组件。技术报告，技术报告，IBM沃森研究公司，2002年。

[4]苹果。XNU项目。https://github .com/apple - oss-distributions/xnu.2024年4月16日通过。

[5]臂。系统MMU支持。https://developer .手臂com/Architectures/System%20MMU%20Suppo rt.2024年4月16日通过。

[6]货车，阿特利达基斯，杰里米·安德鲁斯，罗克萨娜·詹巴苏，
迪米特里斯·米特罗普洛斯和杰森·尼赫。现代操作系统中的POSIX抽象：旧的、新的和缺失的。*第11届欧洲计算机系统会议论文集，欧洲系统16，纽约，美国，2016*。美国计算机协会

[7]自动传感器。自适应平台。https://www .澳大利亚tosar。org/标准/自适应平台。2024年4月16日通过。

[8]，安德鲁·鲍曼，乔纳森·阿帕沃，奥兰·克里格，和蒂莫西罗斯科。在路上的一个叉子上的（）。*《操作系统热点话题研讨会论文集》，HotOS ‘19，第14-22页，纽约，纽约，美国，2019年*。美国计算机协会

[9]安德鲁鲍曼，保罗巴勒姆，埃里斯特甘德，蒂姆哈里斯，丽贝卡艾萨克斯，西蒙彼得，蒂莫西罗斯科，阿德里安舒普巴赫，和阿希里什辛格哈尼亚。多内核：一种针对可扩展多核系统的新操作系统架构。*《ACM SIGOPS第22届操作系统原理研讨会论文集》，SOSP ‘09，第29-44页，纽约，纽约，美国，2009*。美国计算机协会

[10] B.贝尔沙德，T。安德森。拉佐夫斯卡和H。征兵轻量级的远程过程调用。*在第十二届ACM操作系统原理研讨会论文集，SOSP ‘89，第102-113页，纽约，纽约，美国，1989*。美国计算机协会

[11] B.N.贝尔沙德，S。萨维奇。Pardyak, E.G.Sirer, M.E.菲奥钦斯基，D。贝克尔，C。钱伯斯和S。埃格斯在SPIN操作系统中的可扩展性、安全性和性能。*SIGOPS Oper。西斯特。发动机的旋转*, 29(5):267 - 283, dec 1995.

[12]西蒙比格斯，达蒙李，和格诺特海塞瑟。陪审团单片操作系统的设计是有缺陷的：基于微内核的设计提高了安全性。*在第九届亚太系统研讨会的论文集上，APSys ‘18，纽约，纽约，美国，2018年*。美国计算机协会

[13]黑莓。黑莓QNX：针对嵌入式系统的实时操作系统和软件。https://blackb erry.qnx.com/en.2024年4月16日通过。

[14]黑莓。来看看黑莓平板电脑操作系统背后的力量吧。https://www .qnx .网站/公司/一个声明/blackberry_tablet_os。html.2024年4月16日通过。

[15]黑莓。黑莓操作系统的生命的终结。https: //www。黑莓。com/us/en/support/devices /endoflife, 2020.

[16]凯文·布斯，纳米塔，拉姆拉和林钟。一个在操作系统结构与状态管理方面的实验。*在第14届USENIX操作系统设计和实现研讨会上，OSDI 2020，虚拟事件，2020年11月4-6日，第1-19页*。USENIX协会，2020年。

[17]爱德华·布格尼翁，斯科特·迪瓦恩，金舒克·戈维尔，和梅德尔罗森布卢姆。迪斯科：在可伸缩的多处理器上运行商品操作系统。*《ACM计算机系统学报》（TOCS），第15(4)页：412-447页，1997年*。

[18]安东·伯采夫，纳拉亚南，黄永哲，黄开明、谭、杰格。进化的操作系统内核的安全内核驱动程序接口。在麦尔特·施瓦茨科普夫，安德鲁·鲍曼和娜塔查·克鲁克斯，编辑，第19届操作系统热点研讨会，HOTOS 2023，普罗维登斯，美国，2023年6月22-24日，第166-173页。ACM，2023年。

陈[19]，毛延东、王西、董周，尼古拉·泽尔多维奇和M。Frans卡肖克。Linux内核漏洞：最先进的防御和开放的问题。陈海波、张郑、文苏、周媛媛，APSys11亚太系统研讨会，上海，中国，2011年7月11-12日，第5页。ACM，2011年。

崔[20]，松公园和和戎茶。
针对移动设备的图形感知电源控制。*第17届移动系统、应用和服务国际年度会议论文集，MobiSys '19，第469-481页，纽约，美国，2019年*。美国计算机协会

[21]，松公园，
韩哈和霍俊昌。优化手机游戏的能耗。《IEEE关于移动计算的交易》，21（10）：3744-3756,2022年。

[22] CVE.CVE-2021-30769.https://nvd .nist .gov/vuln/detail/CVE202130769.2024年4月16日通过。

[23] CVE.CVE记录。https://cve .米特尔。org/.2024年4月16日通过。

[24] CVEdetails.Linux内核2.6安全漏洞。https://www.cvedetails.com/vulnerability-list/vendor_ id33/product_ id47/version_id410986/LinuxLinuxKernel2 .6 .html.2024年4月16日通过。

[25]·内森·多滕哈恩，卡桑帕利斯，威尔·迪-埃茨，约翰·克里斯韦尔和维克拉姆·艾德夫。嵌套内核：一种用于内核内特权分离的操作系统体系结构。*在第二十届编程语言和操作系统体系结构支持国际会议记录，ASPLOS '15，第191-206页，纽约，纽约，美国，2015*。美国计算机协会

[26]杰夫Dike。用户模式Linux。*在第五届年度Linux展示和会议（ALS 01），奥克兰，CA，2001年11月*。USENIX协会。

[27]比约恩·多贝尔。内存、IPC和L4Re。https://os .inf .tudresden .de/~doebel/downloads/02-MemoryAndIPC .pdf, 2012.

[28]东杜、华志超、夏玉斌、藏
陈海波。XPC：对安全和高效的跨进程调用的架构支持。在斯里拉塔，博比·曼内，希勒里C。亨特和埃里克。阿尔特曼，编辑，
第46届计算机架构国际研讨会论文集，ISCA 2019，美国，凤凰城，2019年6月22-26日，第671-684页。ACM，2019年。

[29]大卫·埃德尔松。在Scal上提供一个LinuxAPI-可K42内核。*在2003年USENIX年度技术会议（USENIX ATC 03），圣安东尼奥，德克萨斯州，2003年6月*。USENIX协会。

[30]，凯文·埃尔芬斯顿和格诺特·海瑟。从L3到seL4我们在20年的L4微粒研究中学到了什么？在迈克尔·卡明斯基和迈克·达林书中，ACM的编辑
SIGOPS第24届操作系统原理研讨会，SOSP '13，法明顿，法法，美国，2013年11月3-6日，第133-150页。ACM，2013年。

[31]道森R。英格勒，M。弗兰斯·卡沙克和詹姆斯W.O'Toole Jr.外内核：一种用于应用程序级资源管理的操作系统体系结构。在迈克尔B。*琼斯，编辑，第十五届ACM操作系统原理研讨会论文集，SOSP 1995，铜山度假村，
美国科罗拉多州，1995年12月3-6日，第251-266页*。ACM，1995年。

[32]布莱恩·福特，戈德玛回来，格雷格·本森，杰伊·勒普劳，
林和奥林浑身发抖。通量OSKit：内核和语言研究的基底。*第16届ACM操作系统原理研讨会，第38,1997-51页*。

[33]布莱恩·福特和杰伊·勒普劳。进化马赫数3.0到迁移线程模型。*在USENIX 1994年冬季技术会议（USENIX 1994年冬季技术会议），旧金山，加利福尼亚州，1994年1月*。USENIX协会。

[34]免费软件基金会。关于GNU许可证的常见问题。https://www .gnu .org/licenses/gplfaq .en。html#MereAggrega tion.2024年4月16日通过。

[35]开放基础。开放和谐项目。https://gitee.文档，和谐，概述。md.2024年4月16日通过。

[36] FreeBSD.FreeBSD中的Linux仿真。https://文档。freebsd.org/en/articles/linux-emulation/.2024年4月16日通过。

[37]谷歌紫红色。Zircon把手。https://fuchsia.dev/fuchsia-src/concepts/kernel/handles.2024年4月16日通过。

[38] Vinod·加纳帕西，马修·J。Renzelmann, Arini Bal-
阿克里希南，迈克尔M。斯威夫特和索梅什Jha。微驱动程序的设计和实现。在苏珊J。埃格斯和詹姆斯R。*Larus，编辑，第13届编程语言和操作系统架构支持国际会议论文集，ASPLOS 2008，西雅图，华盛顿州，2008年3月1-5日，第168-178页*。ACM，2008年。

[39] Alain格夫劳特，特伦特·杰格尔，约野浩公园，约亨
利特克，凯文J。埃尔芬斯顿，乌利格，乔纳森E。蒂德斯韦尔，卢克·戴勒和拉尔斯·鲁瑟。SawMill多服务器方法。*在第九届ACM SIGOPS欧洲研讨会上*

*研讨会：《超越PC：操作系统的新挑战》，EW 9，第109-114页，纽约，纽约，美国，2000年。*美国计算机协会

[40]托马斯·格莱克斯纳和英戈·Molnar。Linux通用IRQ处理。https://www .核心org/doc/html/v4 .18/coreapi/genericirq .html， 2010.

[41]山东努戈尔和丹·杜尚。Linux设备驱动程序模拟，马赫数。*在USENIX年度技术会议上，第65-74页，1996年。*

[42]谷歌。安卓开源项目。https://so urce.机器人。com/.2024年4月16日通过。

[43]谷歌。AOSP兼容性测试套件。https:// 根源机器人。com/docs/compatibility/cts.2024年4月16日通过。

[44]谷歌。AOSP供应商测试套件（VTS）和基础设施。https://source .机器人。com/docs/co re/tests/vts.2024年4月16日通过。

[45]谷歌。应用程序启动时间。https://developer .一机器人比赛主题，表演，重要时刻。2024年4月16日通过。

[46]谷歌。紫红色锆石内核。https://fuchsi a.dev/fuchsia-src/concepts/kernel?hl=en.2024年4月16日通过。

[47]谷歌。紫红色libc。https://fuchsia .dev/fu chsiasrc/development/languages/ccpp/lib c?hl=en.2024年4月16日通过。

[48]间谍杜拉格拉瓦尼，穆罕默德赫达亚蒂，约翰克里斯韦尔和迈克尔L。斯科特使用IskiOS的快速内核内隔离和安全保护。*在第24届攻击、入侵和防御研究国际研讨会的论文集上，RAID '21，第119-134页，纽约，纽约，美国，2021年。*美国计算机协会

顾[49]，吴欣悦，李文泰，刘年，米泽宇，夏玉斌、陈海波。用有效的内核内隔离和通信来协调微内核中的性能和隔离。在阿达·加夫里洛夫斯卡和埃雷兹·扎多克，编辑，2020年USENIX年度技术会议，USENIX ATC2020,2020年7月15-17日，第401-417页。USENIX协会，2020年。

[50]尼古拉汉普顿。正在工作的死机：过时的Linux内核的安全风险。https://www2 .com puterworld.com.au/article/615338/working -deadsecurityriskdatedlinuxkernels/， 2017.

[51]尼古拉·汉普顿和帕特里克·谢奇克。一种分析soho路由器固件货币的调查和方法。2015.

[52] Gernot·海瑟和本·莱斯利。OKL4微监控程序：微内核和管理监控程序的收敛点。在钱德拉莫汉A。周志东，编辑，亚太系统研讨会，2010年，印度新德里，2010年8月30日，第19-24页。ACM，2010年。

[53]令状N。赫尔德，赫伯特博斯，本格拉斯，菲利普霍姆-伯格和安德鲁。塔嫩鲍姆MINIX 3：一个高度可靠的，自修复的操作系统。*SIGOPS Oper。西斯特。发动机的旋转*, 40(3):80 - 89, jul 2006.

[54]丹希尔德布兰德。QNX的架构概述。*在关于微内核和其他内核架构的USENIX研讨会上，第113-126页，1992年。*

[55]汉斯·霍尔伯格和乌多·布罗克迈耶。在基于模型的软件开发过程中，进行符合ISO 26262标准的功能需求验证。*在白皮书中。嵌入式世界展览和会议*，2011.

[56]，黄永哲，维克拉姆·纳拉亚南，大卫·德特韦勒，黄开明、谭、杰格、伯采夫。KSplit：自动进行设备驱动程序隔离。在马科斯K。*阿奎莱拉和哈基姆·威瑟斯彭，编辑，第16届USENIX操作系统设计与实现研讨会，OSDI 2022，卡尔斯巴德，美国，2022年7月11-13日，第613-631页。*USENIX协会，2022年。

[57] 华为 .麒麟9000。https://www .hisilicon .com/en/products/Kirin/Kirinflagshipchips/Kirin9000.2024年4月16日通过。

[58] Jack Tigar，汉弗莱斯，尼尔·纳图，阿什温·肖古勒，韦斯、巴雷特罗登、乔什唐、路易吉里佐、奥列格龙巴克、保罗特纳和克里斯托斯科兹拉基斯。快速和灵活的Linux调度的用户空间委托。在罗伯特范雷内斯和尼科莱泽尔多维奇，编辑，SOSP '21： ACM SIGOPS 28 *操作系统原理研讨会，虚拟事件/科布伦茨，德国，2021年10月26-29日，第588-604页。*ACM，2021年。

[59]盖伦C。亨特和詹姆斯R。鸥属奇点：重新思考软件堆栈。*SIGOPS Oper。西斯特。发动机的旋转*, 41(2):37 - 49, apr 2007.

[60] R英特尔。体系结构指令集的扩展和未来特性的编程参考。https://www .
英特尔。内容内容、开发、外部、美国、文档、架构教学、教学集、扩展、程序参考。pdf, 2021.

[61]等。道路车辆功能安全。https://www .iso.org/obp/ui/#iso: std: iso:26262:-3:ed-1: v1:en.2024年4月16日通过。

[62]等。ISO/IEC 15408-1：2022：信息安全、网络安全和隐私保护——IT安全的评估标准。https://ccsp .阿卢科斯。com/standa rds/isoiec154081- 2022/, 2022.

张秀[63]和姜姜。在过程中
使用硬件监视进行内存隔离。在
第56届年度设计自动化会议论文集，DAC 2019，美国内华达州拉斯维加斯，2019年6月02-06日，第32页。ACM，2019年。

[64]罗伯特·凯泽和斯蒂芬·瓦格纳。进化的PikeOS微内核。*在第一次关于嵌入式系统的微内核的国际研讨会上，第50卷，2007年。*

[65] Antti Kantee等人。灵活的操作系统内部结构：任意内核和臀部内核的设计和实现。2012.

[66]·安托万·考夫曼，蒂姆·斯塔姆勒，西蒙·彼得，Naveen Kr。沙玛，阿文德·克里希那穆西和托马斯·安德森。助教： TCP加速作为一个操作系统服务。*在2019年第十四届欧洲系统会议的会议记录中，第19届欧洲系统会议，纽约，纽约，美国，2019年*。美国计算机协会

[67]格温·克莱因，凯文·埃尔芬斯顿，格诺特·海塞瑟，六月
安德罗尼克，大卫。科克、菲利普·德林、达姆米卡·埃尔卡杜维、凯·恩格尔哈特、拉法尔·科兰斯基、迈克尔·诺里什、托马斯·休厄尔、哈维·图奇和西蒙·温伍德。seL4：对操作系统内核的正式验证。在《马修斯和托马斯·E。*安德森，编辑，第22届ACM操作系统原理研讨会集2009，SOSP 2009，大天空，蒙大拿，美国，2009年10月11-14日，第207-220页*。ACM，2009年。

[68] Orran·克里格，马克·奥斯兰德，布莱恩·罗森堡，罗伯特·W·维斯涅夫斯基、吉米·谢尼迪斯、迪尔玛·达·席尔瓦、米查尔·奥斯特罗斯基、乔纳森·阿帕沃、玛丽亚·布特里科、马克·默根等人。K42：构建一个完整的操作系统。ACM SIGOPS操作系统审查，40(4)：133-145,2006。

[69] l4re.L4Re操作系统框架。https: //l4re。org/.2024年4月16日通过。

[70]亚当·拉克辛的斯基，亚历山大·沃格，马库斯·沃尔普，
和赫尔曼哈蒂格。平坦的层次结构调度。*《第十届ACM嵌入式软件国际会议论文集》，EMSOFT '12，第93-102页，纽约，纽约，美国，2012年*。美国计算机协会

[71]雨果·列福伏尔，弗拉德-安德烈·巴多乌，亚历山大·荣格，
斯蒂凡·卢西安·特奥多雷斯库、塞巴斯蒂安·劳奇、费利佩·胡奇、科斯汀·拉西奥和皮埃尔·奥利维尔。FlexOS：实现灵活的操作系统隔离。在巴巴克·法萨菲，迈克尔·费德曼，掸和托马斯·F。韦尼施，编辑，
ASPLOS '22：27届ACM编程语言和操作系统体系结构支持国际会议，瑞士洛桑，2022年2月28日-2022年3月4日，第467-482页。ACM，2022年。

[72]约书亚·勒瓦瑟，乌利格，扬·斯托斯，和Ste-凤扇Gotz。未经修改的设备驱动程序重用和提高系统的可靠性通过虚拟机。在OSDI，2004年，第17-30页。

[73]罗伊·莱文，埃利斯S。科恩，威廉M。科温，弗雷德J。
波拉克和威廉。伍尔夫水螅体中的政策/机制分离。在詹姆斯C。*布朗和胡安·罗德里格斯-罗塞尔，编辑，第五届操作系统原理研讨会论文集，SOSP 1975，美国德克萨斯大学奥斯汀分校，1975年11月19-21日，第132-140页*。ACM，1975年。

[74]亨利M利维。*基于能力的计算机系统。*
数字出版社，2014年。

[75] Jochen利特克。通过内核设计来改进IPC。在安德鲁P。*布莱克和芭芭拉·利斯科夫，编辑，第十四ACM操作系统原理研讨会论文集，SOSP 1993，格罗夫公园酒店和乡村俱乐部，阿什维尔，北卡罗来纳，1993年12月5-8日，第175-188页*。ACM，1993年。

[76] Jochen利特克。在微观内核建设。在迈克尔B。*琼斯，编辑，第十五届ACM操作系统原理研讨会论文集，SOSP 1995，铜山度假村，美国科罗拉多州，1995年12月3-6日，第237-250页*。ACM，1995年。

[77]汉斯·利尔杰斯特兰，托马斯·尼曼，王奎，汽车-洛杉矶中国佩雷斯，扬-埃里克埃克伯格，和N。Asokan。PAC向上：使用ARM指针身份验证实现指针完整性。*发表在第28届USENIX安全研讨会（USENIX安全19页），第177-194页，圣克拉拉，加州，2019年8月*。USENIX协会。

刘[78]静，安东尼·瑞贝罗，戴一凡，叶陈浩，
苏达松·坎南，安德里亚C。阿帕西-杜索和雷姆齐。阿帕奇杜索。在文件系统半微内核中的规模和性能。*在罗伯特·范·雷内斯和编辑，编辑，SOSP '21： ACM SIGOPS第28届操作系统原理研讨会，虚拟事件/科布伦茨，德国，2021年10月26-29日，第819-835页*。ACM，2021年。

[79]埃尔顿Lum。研究证实，微内核在本质上更安全。https://blogs .黑莓。研究证实，在2020年，它天生就更安全。

[80]安娜·莱昂斯，肯特·麦克劳德，赫萨姆·阿拉木图，和格诺特·海瑟。调度上下文功能：一种有原则的、轻便的、管理时间的操作系统机制。*在第十三届欧洲系统会议的会议记录中，欧洲系统会议的第18届，纽约，纽约，美国，2018年*。美国计算机协会

[81] Anil·马德瓦普迪，理查德·莫蒂埃，查拉兰波斯罗特索斯，大卫J。斯科特，巴拉吉·辛格，托马斯·加扎内尔，史蒂文·史密斯，史蒂文·汉德和乔恩·克劳克罗夫特。统一内核：针对云的库操作系统。*在维维克·萨卡尔和拉斯蒂斯拉夫·博迪克，编辑，编程语言和操作系统的架构支持，ASPLOS 2013，休斯顿，美国，2013年3月16-20日，第461-472页*。ACM，2013年。

[82] Linux手册页。VDSO：虚拟动态共享对象。https://man7 .org/linux/man-pages/m an7/vdso .html..72024年4月16日通过。

[83]延东，陈浩刚、董周、王西，
尼古拉·泽尔多维奇和M。Frans卡肖克。具有API完整性和多主模块的软件故障隔离。在泰德·沃伯和彼得·德鲁切尔的编辑作品中，
第23届ACM2011年操作系统原理研讨会论文集，SOSP 2011，Cascais，葡萄牙，2011年10月23-26日，第115-128页。ACM，2011年。

[84]，迈克尔·马蒂，马克·德·克鲁伊夫，雅各布·阿德里安斯，
克里斯托弗·阿尔菲尔德，肖恩·鲍尔，卡洛·康塔瓦利，迈克·道尔顿，南迪塔·杜基帕蒂，威廉·C。埃文斯、史蒂夫·格里布尔、尼古拉斯·基德、罗曼·科诺诺夫、高塔姆·库马尔、卡尔·莫尔、艾米丽·穆西克、莉娜·奥尔森、迈克·瑞安、埃里克·鲁博、凯文·斯普林伯恩、保罗·特纳、瓦拉斯·瓦兰修斯、王西和阿明·瓦达特。Snap：一种面向主机网络的微内核方法。*在ACM SIGOPS第27届操作系统原理研讨会上，纽约，纽约，美国，2019年*。

[85]拉里W。麦克沃伊和卡尔·斯塔林。便携式
性能分析工具。*在诉讼程序
USENIX年度技术会议，圣地亚哥，美国，1996年1月22-26日，第279-294页*。USENIX协会，1996年。

[86]：米泽宇、李定二、杨子涵、王欣然、和
陈海波。针对微内核的快速和安全的进程间通信。*在2019年第十四届欧洲系统会议的会议记录中，第19届欧洲系统会议，纽约，纽约，美国，2019年*。美国计算机协会

[87]微软。用于Linux文档的Windows子系统。https://learn .微软。com/en-us/windows/wsl/.2024年4月16日通过。

[88]微软。NT内核模式用户和GDI白皮书。https://learn .微软。以前的版本/cc750820(v.=技术网。10)，2014.

[89]直到米米茨，马克西姆行星和维克多·劳林
雷斯奇。快速系统调用的新机制。*arXiv预印本arXiv：2112.10106,2021年*。

[90]Vik拉姆·纳拉亚南，阿比拉姆·巴拉苏布拉曼尼亚，查尔-
雅各布森、莎拉·斯波尔、斯科蒂·鲍尔、迈克尔·奎格利、阿夫塔布·侯赛因、阿卜杜拉·尤尼斯、沈俊杰、莫伊纳克·巴塔查里亚和安东·伯采夫。lxd：旨在实现内核子系统的隔离。*在达丽亚·马尔基和丹·特萨夫里尔，编辑，2019 USENIX年度技术大会，USENIX ATC 2019，伦顿，美国，2019年7月10-12日，第269-284页*。USENIX协会，2019年。

[91]维克拉姆·纳拉亚南，黄永哲，刚谭，特伦特
杰格和安东·伯采夫。具有虚拟化和VM功能的轻量级内核隔离。在桑托什
纳加拉卡特，安德鲁·鲍曼和巴里斯·卡西奇，编辑，第20：16届ACM签署计划/SIGOPS虚拟执行环境国际会议，虚拟事件[瑞士洛桑]，2020年3月17日，第157-171页。ACM，2020年。

[92] Ruslan尼古拉耶夫和戈德玛回来了。VirtuOS：一个具有内核虚拟化的操作系统。*在迈克尔·卡明斯基和迈克·达林，编辑，ACM SIGOPS第24届操作系统原理研讨会，SOSP '13，宾夕法尼亚州，美国，2013年11月3-6日，第116-132页*。ACM，2013年。

[93]艾米·奥斯特豪，约书亚·弗里德，乔纳森·贝伦斯，
亚当贝莱，和哈里巴拉克里希南。申南戈：为对延迟敏感的数据中心工作负载实现较高的CPU效率。*在第16届USENIX网络系统设计与实现研讨会上*（NSDI 19），第361-378页，波士顿，马州，2019年2月。USENIX协会。

[94]加布里埃尔帕默。线程迁移的情况是：在一个可定制的和可靠的操作系统中的可预测的ipc。*发表在嵌入式实时应用程序操作系统平台研讨会论文集（OSPERT 2010），第91页，2010年。*

[95]罗马佩尼亚耶夫。epoll：确保准备就绪列表中的所有元素都按FIFO顺序排列。https://patchwork .核心org/project/linuxfsdevel/patch/20181212110357.25656-2-rpenyaev@suse.德，2018年。

[96]，西蒙·彼得，李家林，张艾琳，丹·R。K.端口，道格·沃斯，阿尔文德·克里希那穆蒂，托马斯·安德森和蒂莫西·罗斯科。操作系统是控制平面。*ACM跨。压缩。西斯特。*，33(4)，nov 2015.

[97]肖恩彼得斯，阿德里安丹尼斯，凯文埃尔芬斯顿，和盖尔-
不是海泽。对于一个微内核来说，一个大的锁也可以。*在第六届亚太地区系统研讨会的会议记录中，APSys '15，纽约，纽约，美国，2015年。*美国计算机协会

[98]马克·皮奇福德。在关键应用中使用Linux：比如混合油和水？https://www.嵌入式。com/usinglinux-withcriticalapplicati onslike-mixingoiland-water/，2021.

[99]灵长类动物实验室公司。极客5 CPU工作负载。https://www.geekbench.5的工作负载。pdf，2019.

[100]普罗斯库林，马吕斯·莫乌，塞耶德默德
加瓦尼亚，瓦西里奥斯。克默里斯和米卡利斯。用于内核和用户空间的选择性内存保护。*2020年IEEE安全与隐私研讨会，SP 2020，旧金山，美国，2020年5月18-21日，第563-577页。*2020年IEEE。

[101]屋大维普迪拉，卢西安阿德里安格里金库，和尼古拉塔普斯。LKL：Linux内核库。*第9届RoEduNet IEEE国际会议，第328-333页。*2010年IEEE。

[102] Ali Raza，托马斯·昂格尔，马修·博伊德，埃里克·B
曼森、帕鲁尔·索哈尔、乌尔里希·德雷珀、理查德·琼斯、丹尼尔·布里斯托特·德奥利维拉、拉里·伍德曼、雷纳托·曼库索、乔纳森·阿帕沃和奥兰·克里格。统一内核Linux（UKL）。*在第十八届欧洲计算机系统会议的会议记录中，欧洲计算机系统会议的第23页，第590-605页，纽约，纽约，美国，2023年。*美国计算机协会

[103]项（珍妮）任，柯克·罗德里格斯、陈绿源，
卡米洛维加，斯图姆，和丁源。Linux核心系统的性能演变分析
操作*发表在第27届ACM操作系统原理研讨会论文集中，SOSP '19，第554-569页，纽约，纽约，美国，2019年。*美国计算机协会

[104]蒂莫西罗斯科。是时候使用操作系统了
重新发现硬件。USENIX协会，2021年7月。

[105] Leonid Ryzhyk，彼得·丘布，伊霍尔·库兹，和格诺特
海泽驯服设备驱动程序。在沃尔夫冈·施罗德-普雷克沙特，约翰·威尔克斯和丽贝卡·艾萨克斯，编辑，2009年欧洲系统会议论文集，德国纽伦堡，2009年4月1-3日，第275-288页。ACM，2009年。

[106] Leonid Ryzhyk，彼得·丘布，伊霍尔·库兹，艾蒂安·勒
Sueur和Gernot Heiser。与白蚁有关的自动装置驱动器合成。在《马修斯和托马斯·E。*安德森，编辑，第22届ACM操作系统原理研讨会集，2009，SOSP 2009，Big Sky，蒙大拿，美国，2009年10月11-14日，第73-86页。*ACM，2009年。

[107] seL4.seL4功能。https://docs .sel4 .systems/Tutorials/capabilities .html.2024年4月16日通过。

[108] Arvind塞沙德里，马克鲁，曲宁，和阿德里安佩里格。SecVisor：一个小型管理程序，为商品操作系统提供终身内核代码完整性。在托马斯C。布雷索德和M。*弗兰斯，编辑，第21届ACM操作系统原理研讨会2007，SOSP 2007，史蒂文森，华盛顿，美国，2007年10月14-17日，第335-350页。*ACM，2007年。

[109]山益州、黄宇通、陈依伦、一英
张。LegoOS：一种用于硬件资源分解的分散的分布式操作系统。*在第13届USENIX操作系统设计与实现研讨会（OSDI 18）上，第69-87页，卡尔斯巴德，加州，2018年10月。*USENIX协会。

[110] Anand Lal Shimpi。黑莓剧本评论。https://www.anandtech.com/show/4266/blackberry-playbookreview/14，2011.

[111] Dilma Da席尔瓦，奥兰·克里格，罗伯特·W。维希涅夫斯基
阿莫斯沃特兰，大卫塔姆，和安德鲁鲍曼。K42：一个用于操作系统研究的基础设施。*SIGOPS Oper。西斯特。发动机的旋转*，40(2):34 - 42，apr 2006.

[112]迈克尔M。斯威夫特，布莱恩。伯沙德和亨利M。征兵提高大宗商品操作系统的可靠性。*SIGOPS Oper。西斯特。发动机的旋转*，37(5):207 - 222，oct 2003.

[113] Sysgo.开源和ASIL D认证-可能吗？https://www .sysgo .2018年可能认证。

[114]威利塔罗。Linux 2.6.32.71（EOL）。https://lwn .net/Articles/679874/, 2016.

[115]的Linux内核文档。能源意识调度。https://docs .核心org/scheduler/分裂能源。html, 2019.

[116]Chia-蔡，布山耆那，纳菲·艾哈迈德·阿卜杜勒，和唐纳德E。看门人对现代Linux API的使用和兼容性的研究：当您支持时应该支持什么。*第11届欧洲计算机系统会议论文集，欧洲系统16，纽约，美国，2016*。美国计算机协会

[117]约翰内斯和雷克。家庭路由器安全报告2022年。https://www .fkie .令人担忧的人。de/content/dam/fkie/de/documents/20221128_HRSR_2022 .pdf, 2022.

[118] Hannes·韦斯巴赫、比约恩·多贝尔和亚当·拉克钦斯基。通用用户级PCI驱动程序。*在第13届实时Linux研讨会的论文集中。*, 2011.

[119]嵌入式Linux Wiki。嵌入式Linux系统大小。https://elinux.组织/系统大小. 2024年4月16日通过。

[120]艾米特·威切尔，李俊万和阿萨诺维。蒙德里克斯：使用蒙德里安内存保护的Linux内存隔离。*在第二十届ACM操作系统原理研讨会的论文集中，SOSP '05，第31-44页，纽约，纽约，美国，2005年*。美国计算机协会

[121] Hyrum赖特。Hyrum定律。https://www .hyrumslaw.com/, 2012.

吴方诺[122]、董明凯、莫格泉、海波陈。TreeSLS：一个在NVM上具有树状结构状态检查点的全系统持久性微核。在*第29届操作系统原理研讨会论文集，SOSP '23，第1-16页，纽约，纽约，美国，2023年*。美国计算机协会

周冯[123]，杰里米·康迪特，扎卡里·R。安德森，伊利亚巴格拉克，罗伯特·恩纳尔斯，马修·哈伦，乔治·C。Necula和Eric A。啤酒商。使用基于语言的技术进行安全和可恢复的扩展。在布莱恩N。伯沙德和杰弗里C。*莫卧儿，编辑，第七届操作系统设计研讨会和实施OSDI '06,11月6-8日，西雅图，美国，第45-60页*。USENIX协会，2006年。