开题报告

QvQ

目录

小组成员

- 楚颖 PB20111610
- 耿双越 PB20111639
- 郝英屹 PB20050920
- 隋国冲 PB20111643
- 万方 PB20111645

> 按姓氏首字母顺序排列，排名不分先后。

项目概述

RedLeaf 是用 Rust 从 0 开始开发的微内核，旨在探索语言安全对操作系统组织的影响。本项目将以同样的设计思路，在 ARM 上实现一微内核，并基于此语言隔离机制开发相关内核模块，验证在不同架构下其设计的可行性与运行开销。 我们希望，本项目作为一次对基于语言的隔离机制的探索，尤其能够为低功耗，资源受限的平台提供一些参考。
我们将首先在模拟器中验证设计思路，如果有机会我们也会直接在树莓派上运行我们的设计，并以 RedLeaf 模块形式开发部分简单外设驱动，检验该设计。

项目背景

跨架构迁移的机遇与挑战

背景

在计算机体系结构的发展过程中，诞生了 CISC (复杂指令集)和 RISC (精简指令集)这两大流派。基于这两类指令集，产生了两种主流的 CPU 架构： x86 架构，采用的是 CISC 复杂指令集，而 ARM 架构则采用了 RISC 精简指令集。两种架构各有优缺点。x86 结构的系统在性能和兼容性方面比 ARM 结构的系统强得多，但是其效率比较低，功耗与 ARM 结构的电脑无法相比。相反，ARM 最大的优势就在于功耗。Rust 语言标准库是让 Rust 语言开发的软件具备可移植性的基础。在裸金属环境下，Rust 可以编译启动代码、内核等。

挑战

目前跨平台移植操作系统有以下几个问题。首先，内核中通常有一小部分是使用汇编语言完成的，例如启动代码等，若要完成这一部分的移植，就只能使用另一种汇编语言重写；第二，有个别 Bug ，在 x86 上和 ARM 上的表现不同，例如 ARM 和 x86 调用函数的方式，参数传递入栈的方式不一样，有些栈溢出问题在 x86 上没有表现，却会导致 ARM 的崩溃；第三，性能问题，x86 的性能高于 ARM ，因此完成移植之后需要进行性能调优，移植后系统的瓶颈通常会在 CPU 过于繁忙、IO 等待、网络等待、响应时延等方面出现，此时需要对瓶颈点实施进一步调优策略。

现有解决方案

编译器能够依照如下顺序，将高级语言翻译成二进制码：高级语言->汇编语言->二进制机器码。现在的编译器往往已经相当成熟，对于高级语言的部分，很多问题都可以由编译器自动解决。

硬件抽象层（HAL，Hardware Abstraction Layer），是位于操作系统、内核与硬件电路之间的接口层，其目的在于将硬件抽象化。它隐藏了特定平台的硬件接口细节,为操作系统提供虚拟硬件平台,使其具有硬件无关性,可在多种平台上进行移植。而在我们即将使用的 Rust 语言提供更加高效的嵌入式硬件抽象层，能够在完成基本功能的前提下增加复用性，从而降低复杂度。

意义

在当前数字产业大发展的背景下， 对 ARM 架构的关注必然会不断升温，各行业会出现大量应用从 x86 向 AMR 架构迁移，也催生了大量的应用跨架构适配测试需求。我们将在 ARM 上完成微内核，验证在不同架构下其设计可行性与运行开销，希望能够为今后的工作提供参考。

细粒度隔离与性能的冲突

现代操作系统的内核往往格外庞大，难免出现局部 Bug，需要良好的隔离机制。但同时，在实践中，我们不能忽视因此带来的性能开销，寻求高效的实现方式是解决此冲突的核心。

硬件实现

- 硬件实现难以保证跨平台通用性，且通常粒度较粗。因此,基于硬件实现的隔离机制往往要么隔离不充分，如宏内核，各种驱动程序，文件系统等功能均运行在内核空间，牵一发而动全身，直接增加了内核开发成本；要么性能不理想，如饱受 IPC 性能制约的微内核。
- 也存在一些使用虚拟化硬件做细粒度隔离的尝试，但他们的功能依然受限于特定指令集，且实现较为复杂。
- 近年更新的硬件技术也在推进更精细的隔离，如 Intel 的 PKS 技术用于加强内核态的内存保护，但要达到完美仍需语言隔离配合。

语言实现

- 使用带有垃圾回收机制的语言，如 go，可以较为容易地做到良好地隔离设计，但因此带来的运行时开销不容忽视。
- Rust 通过严格的所有权机制等特性，无需垃圾回收机制就能提供语言上的安全性，通过允许内核内部分 unsafe 代码，并限制内核模块间接口行为, 我们能使用 Rust 开发出一种微内核，其既能继承微内核的模块化设计，保证安全隔离，又能在经过编译器检查后编译到同一地址空间，免除上下文频繁切换的缺点，达到接近宏内核的性能。这正是我们的目的所在。

现有解决方案

Lightweight Virtualized Domains

LVDs 使用 VT-x 指令集引入的硬件辅助虚拟化技术，为宏内核提供了一种隔离的新思路。
LVDs 使用多个 EPT 运行不同的内核模块，使用 VMFUNC 切换页表进行跨域调用，仅引入了相当于系统调用的开销就能实现良好的保护机制。为现有宏内核的模块化提供了一种思路。

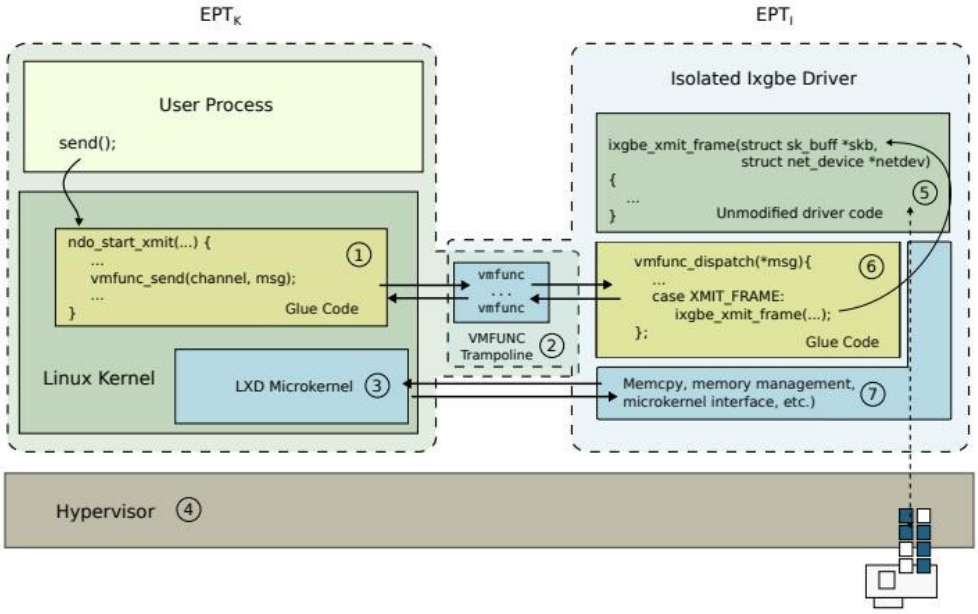


Figure 1. LVDs architecture.

意义

目前的隔离机制主要有硬件隔离和语言隔离。Rust 通过严格的所有权机制等特性，不需垃圾回收机制就能提供语言上的安全性，因而避免了大部分运行时的开销。额外的开销主要在于边界检查，而边界检查对于现代CPU来说往往可以通过分支预测避免性能损失。

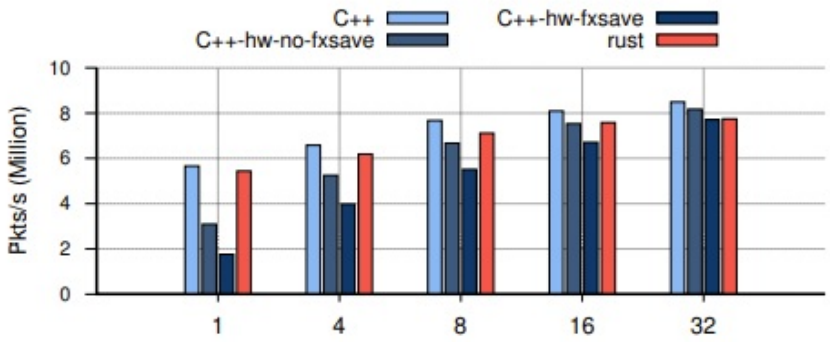


Figure 2. Isolation overheads on varying batch sizes

微内核设计的优势与不足

背景

如上文所述，由于仅提供基本的进程调度，内存管理和中断分配等功能，在安全与稳定上，微内核天然具有优势，同时也可以通过 libOS 的形式方便的扩展出精准匹配自身需求的服务和驱动。
但微内核的优雅设计也带来了性能的折扣，这一点尤其体现在频繁 IPC 带来的开销上。

优势：

- 微内核架构很小且是隔离的，因此可以更好地运行。
- 包含的功能很少有涉及破坏性的风险，因而安全性和稳定性很好。
- 服务器的故障也像其他用户程序的故障一样被隔离。
- 微内核系统是灵活的，不同的策略和API由不同的服务器实现，它们可以在系统中共存。
- 系统的扩展更容易获得，因此可以在不干扰内核的情况下将其添加到系统应用中。
- 安全性和稳定性的提高将导致在内核模式下运行的代码量减少
- 微内核是模块化的，不同的模块可以被替换、重新加载、修改，甚至不需要触及内核；无需重新

编译，就可以添加新的功能。

不足：

- 与普通的宏内核系统相比，在微内核系统中提供服务的成本较高。
- 当驱动分别被实现为程序或进程时，需要上下文切换或函数调用。
- 微内核系统的性能有所降低，可能会引发一些问题。

现有解决方案

SkyBridge

SkyBridge 使用虚拟化技术(VT-x)，在微内核(Subkernel)下层插入一个 Rootkernel 作为虚拟化层，IPC 不经过微内核，直接无缝切换到另一个进程的内存空间，在常见的微内核中取得了可观的性能提升，包括 IPC 性能作为业界顶级的 seL4 (IPC 速度提升 149%, 吞吐量提升 81.9%)

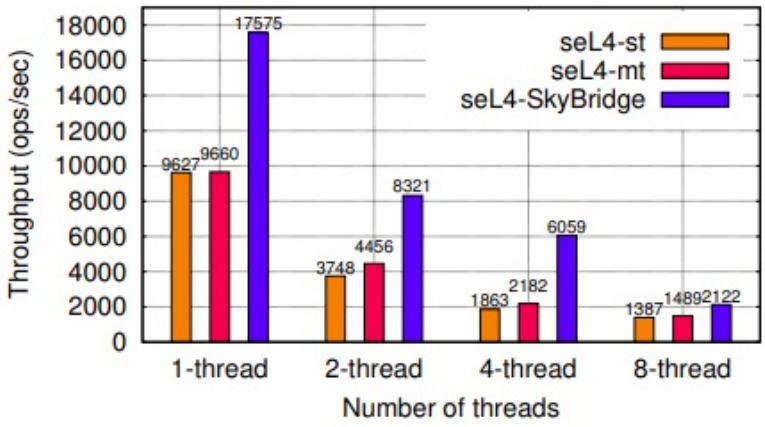


Figure 9. The throughput of YCSB-A (in op/s) for seL4. Higher is better.

嵌入式系统的安全需求

背景

鉴于嵌入式系统资源严重受限且对安全性和稳定性的要求进一步提高，上述的矛盾在其中更加突出。现有的小型嵌入式系统内核常常不支持模块隔离，其所有的任务运行在同一内存空间，这种做法有助于节省本就有限的系统资源，但缺点是每个被操作系统调度的任务均有可能出错，而由于无法将单个任务崩溃造成的影响控制在此任务之内，整个系统的安全性和稳定性都会随之降低。

意义

用 Rust 开发内核，能够在提高性能和安全性的同时不添加额外的资源开销、保持高效率，RedLeaf 的优秀的内核模块隔离机制，在未来，尤其是在资源受限的平台上，会有相当大的潜力。同时，模块的自由组合可以提供高度定制化的能力，面向嵌入式多种多样的硬件和业务需求，我们的工作有潜力成为一个安全稳定的开发平台。我们希望能填补该方向的空白，为今后研究该方向提供一点参考。

嵌入式系统的产业化需求

背景

嵌入式系统的应用十分广泛，涉及工业生产日常生活、工业控制、航空航天等多个领域，而且随着电子技术和计算机软件技术的发展，不仅在这些领域中的应用越来越深入，而且在其他传统的非信息类设备中也逐渐显现出其用武之地。目前嵌入式系统的市场规模正在迅速扩大，在未来的几年里其复合年均增长率预计超过 6%，越来越多的产品被创造出来并投入使用。将来的“万物互联”更是会促进嵌入式系统的发展。与此同时，各行各业对提升操作系统性能、安全性和低功耗的要求越来越高。将来的许多系统可能要 24*7 小时不间断工作。为了满足市场的需求，新出现的操作系统必须有更好的性能、更高的安全性和更低的功耗。

意义

高效又稳定的操作系统能够方便各种物联网设备的快速接入，更好地支撑智慧城市、智能水表、智能家居、智能穿戴、车联网等多种行业应用。

立项依据

如前所说，现代操作系统已经可以建立良好的隔离机制，主要是硬件隔离和语言隔离。硬件隔离原语的成本很高，往往伴随着跨平台的障碍和较粗粒度的限制，其安全性也有所欠缺；语言隔离可以实现更精确、粒度更小的隔离机制，也可以实现较大的安全性，但是带来的开销也是可观的。因此，我们希望可以探索出一种实现安全隔离同时开销较小、性能稳定的隔离机制应用到微内核上。

系统的可靠性可以宽泛地表述为一个“正常工作”的系统不会出现其设计者、开发者、管理者或用户意料之外的行为，目前的系统缺乏可靠性，最明显的表现是容易出现意料之外的安全漏洞。自身不稳定、第三方代码导致的不稳定性、软件安装更新或移除时的不确定行为，都是当前系统缺乏可靠性的最明显表现。
今天，操作系统内核内部缺乏隔离是破坏其安全性的主要因素之一。而 Redleaf 提供了良好的模块隔离机制，背离昂贵的硬件要求，仅使用了 Rust 语言的类型和内存安全性，是探索细粒度和轻量级的隔离的成功案例。我们可以学习 Redleaf 的设计模式，为本项目微内核的实现提供了思路。

Rust 丰富的类型系统和所有权模型保证了内存安全和线程安全，同时不添加额外的资源开销，保持轻量化，是当前最适合接替 C/C++ 来实现本项目所追求的微内核的语言。

同时也启发了一种优秀的内存分配方案：微内核记录分配大的内存区域（记录在 heap registry ）；为了在私有堆中提供细粒度的内存分配，每个域都和一个 RUST 内存分配接口。这样效率和安全性就都具备了。

为了探索微内核在资源受限平台的应用性，针对两种主流的 CPU 架构，考虑在 ARM 上实现我们的微内核，同时进行性能调优，这也与当今应用跨架构适配测试需求不谋而合。

一个功能齐全的微内核（支持网络驱动、NVMe 等）也越来越受重视，这对于跨设备部署等问题的解决意义重大。由 Singularity OS 启发，Redleaf 对隔离机制的探索，提供了细粒度访问控制、透明恢复、最低特权等的实践思路。

QvQ 的初步架构是掌握 Redleaf 的语言隔离模式，利用 Rust 语言特性设计出一个安全稳定且轻量级的微内核。若时间充裕则考虑做出跨架构平台的设计，将其移植到树莓派上。

相关工作

Windows on Arm

目前的 PC 架构绝大多数都是 Intel 的 x86 架构，但是随着人们对设备功耗的要求逐渐升高，近些年来已经有一些 PC 操作系统向 ARM 架构移植。Windows on Arm 最初于 2017 年底推出，提供典型的 Microsoft 操作系统体验，但在 ARM 处理硬件上运行。与传统 Windows 笔记本电脑相比，它的价值主张是承诺提供超过一天的超长电池寿命、通过 4G 和 5G 始终在线的互联网连接、超快速启动以及对 Windows Hello 等安全功能的芯片组级支持。但是，有一些平台警告，包括缺乏游戏支持、偶尔出现的驱动程序问题以及不能完全涵盖可能需要的所有内容的软件仿真。从更长的电池寿命到更薄和无风扇的设计，这对消费产品有重大影响。基于 ARM 的笔记本电脑正越来越多地进入主流意识。

嵌入式 OS

TencentOS tiny

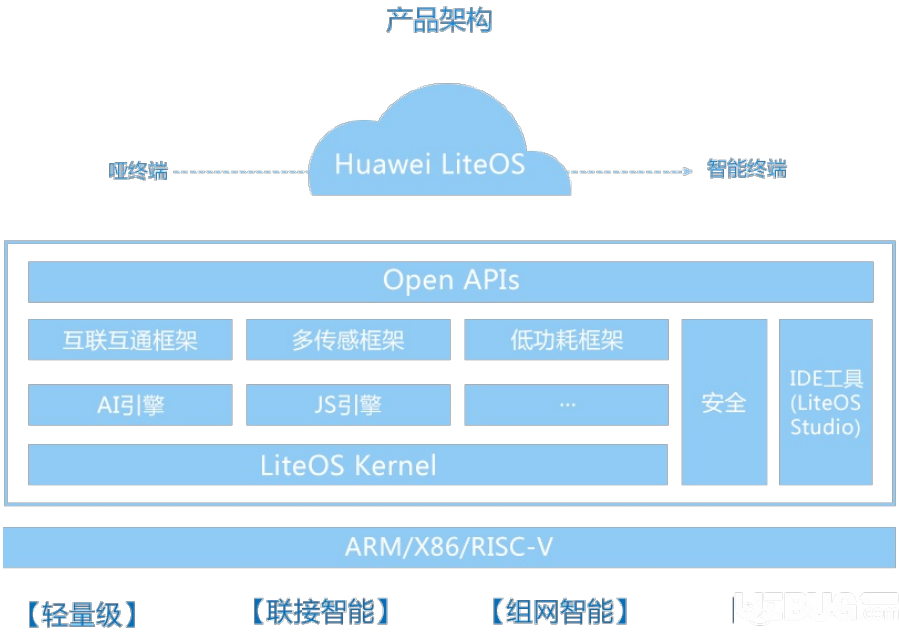
TencentOS tiny是腾讯面向物联网领域开发的实时操作系统，具有低功耗，低资源占用，模块化，安全可靠等特点，可有效提升物联网终端产品开发效率。TencentOS tiny 提供精简的 RTOS 内核，内核组件可裁剪可配置，可快速移植到多种主流 MCU (如 STM32 全系列)及模组芯片上。而且基于 RTOS 内核提供了丰富的物联网组件，内部集成主流物联网协议栈（如 CoAP / MQTT / TLS / DTLS / LoRaWAN / NB-IoT 等）。

优势

- 小体积：最小内核RAM 0.6KB，ROM 1.8KB；典型 LoraWAN 及传感器应用：RAM 3.3KB，ROM 12KB。
- 低功耗：休眠最低功耗低至2 uA；支持外设功耗管理框架。
- 可靠的安全框架：多样化的安全分级方案；均衡安全需求 & 成本控制。
- 良好的可移植性：内核及IoT组件高度解耦，提供标准适配层；提供自动化移植工具，提升开发效率。

Huawei LiteOS

Huawei LiteOS 是华为面向物联网领域开发的一个基于实时内核的轻量级操作系统。现有基础内核包括不可裁剪的极小内核和可裁剪的其他模块。极小内核包含任务管理、内存管理、异常管理、系统时钟和中断管理。可裁剪模块包括信号量、互斥锁、队列管理、事件管理、软件定时器等。除了基础内核，Huawei LiteOS 还提供了扩展内核，包括 C++ 支持、动态加载、低功耗以及维测模块。低功耗通过支持 Tickless 机制、run-stop 休眠唤醒，可以大大降低系统功耗。维测部分包含了获取 CPU 占用率、支持串口执行 Shell 命令等功能。



Contiki

Contiki 最初是作为 WSN 的操作系统开发的，运行在内存非常有限的 8 位 MCU 上，但现在也可以运行在 16 位 MCU 和基于 ARM 32 位 MCU 的现代 IoT 设备上。它是基于事件驱动的合作调度方法，支持轻量级的伪线程。虽然是用 C 语言编写的，但操作系统的某些部分利用了基于宏的抽象（例如

Protothreads），实际上要求开发者考虑某些限制，即他们可以使用什么类型的语言功能。Contiki有几个网络堆栈，包括流行的 uIP 堆栈，支持 IPv6、6LoWPAN、RPL 和 CoAP；以及 Rime 堆栈，它提供了一套分布式编程抽象。

RIOT

RIOT 是多线程操作系统类别中较为突出的操作系统。它旨在建立一个对开发者友好的编程模型和 API，例如类似于Linux 上的经验。RIOT 是一个基于微内核的实时操作系统，支持多线程，使用继承自 FireKernel 的架构。虽然该操作系统是用C语言（ANSI99）编写的，但应用程序和库也可以用 C++ 实现。

FreeRTOS

FreeRTOS 是一个流行的实时操作系统，已被移植到许多 MCU。它的抢占式微内核支持多线程。它采用的是修改过的GPL，没有提供自己的网络堆栈，但第三方网络堆栈可以用于互联网连接。

ThreadX

ThreadX 是一个实时操作系统，基于一个微内核 RTOS（有时被称为 picokernel），它支持多线程并使用一个抢占式调度器。内核提供了两种技术来消除优先级反转。

- 优先级继承，在执行关键部分时提升任务的优先级；
- 抢占阈值，禁止抢占低于指定优先级的线程。
- 其他功能，如网络堆栈、USB 支持、文件系统或 GUI，可以作为单独的产品购买。

QNX

QNX 是基于微内核的实时操作系统之一，并提供了类似 UNIX 的 API。目前的版本名为 QNX Neutrino，支持众多架构，但没有一个符合1类设备的要求。

VxWorks

VxWorks 是一个单片机内核，主要支持 ARM 平台和英特尔平台，包括新的 Quark SoC。VxWorks 支持 IPv6 和其他物联网功能，但缺乏对 6LoWPAN 协议栈的支持，无法适用于 RFC 7228 所定义的受限物联网设备。

树莓派 **OS** 参考

树莓派是 ARM 架构的卡片电脑，本项目在最终有能力的情况下将尝试将实现的微内核移植到树莓派上。

Raspbian

Raspbian 是专门用于 ARM 卡片式计算机 Raspberry Pi “树莓派”的操作系统。Raspbian 系统是 Debian 7.0/wheezy 的定制版本。得益于 Debian 从 7.0/wheezy 开始引入的“带硬件浮点加速的 ARM 架构”(armhf)，Debian 7.0 在树莓派上的运行性能有了很大提升。Raspbian 默认使用 LXDE 桌面，内置 C 和 Python 编译器。

PiNet

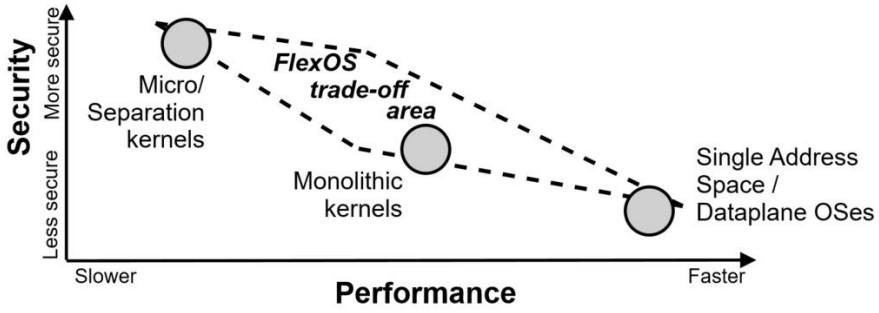
PiNet 是一个自由和开放源码的项目，为帮助学校建立和管理一个 Raspberry Pi 的课堂。其主要特点包括：

1. 基于网络的用户帐户
2. 基于网络的操作系统 - 所有树莓 PIS 启动一个主 Raspbian 操作系统
3. 共享文件夹 - 易于使用共享文件夹系统的教师和学生
4. 工作收集系统简单的工作收集和提交系统，让学生在工作
5. 自动备份 - 自动备份所有学生的工作，定期向外部驱动器
6. 多个小的功能，如批量用户导入，课堂管理软件集成等

harmony-raspberry

harmony-raspberry 项目，主要研究将鸿蒙 Harmony 1.0 移植到树莓派 raspberry Pi 2B，同时，树莓派 raspberry Pi 4B 正在移植中。当然，目前移植到树莓派上的并不是完整版的 HarmonyOS，很多树莓派的驱动还没有实现，只是进入了 Shell 命令行界面。

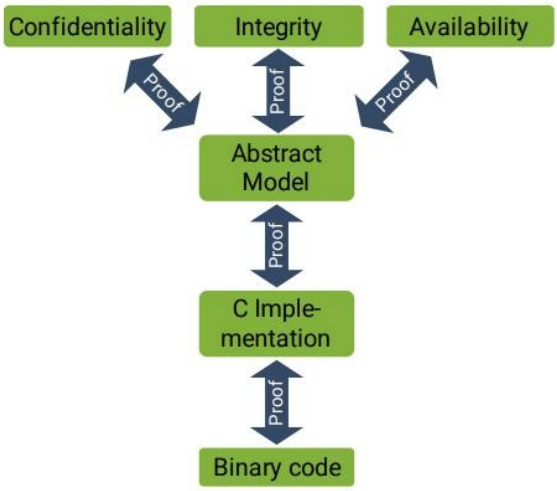
FlexOS



- 硬件实现
- 运行时检查
- 形式化验证

其中每一种都有各自的优势，但也存在各自的问题。如硬件面临的旁路攻击，其中著名的 MetIdown 漏洞通过 CPU Cache 预取暴露被保护数据。
同时，当今越来越多的任务需要多种保护机制结合使用。
但目前，OS 所采用的保护机制往往在设计阶段确定，开发完成后，如果要进行定制，则需重构大量代码，FlexOS 针对这种低效的开发流程提出了一种新的设计理念，在编译 OS 时根据模块的需要，从多种模板中选出一种展开为具体的实现。
使用者可以根据自身需求，在无需改动代码的前提下选择不同保护机制实现方式，从而减少了不必要的重复开发，提高了定制性和灵活度。 同时也有助于在各种实现的不同组合中寻找最佳的 性能-安全性-开发周期 平衡点。

seL4



正确性

- 功能正确
  将 C 限制在一个良好定义的子集内，从而通过 C 解析器转换为数学逻辑进行验证，确保 C 代码没有缺陷。
- 编译验证
  为防止使用有缺陷或恶意的编译器，另外验证编译器和链接器生成的可执行二进制文件，证明二进制代码是 C 代码的正确翻译。

Capability

- 细粒度访问控制
  能力提供细粒度的访问控制，符合最小权限安全原则，相比之下更传统的访问控制列表方式是一种非常粗粒度的方式。 能力提供了面向对象的访问控制形式，如需要一个不受信任的程序处理一

个文件，我们可以将一个能力交给程序允许读写一个文件，这个程序不能访问任何其他事物。

- 中介与授权
  一个程序被赋予了一个对象的能力，但它只能调用这个对象上的方法，而不知道这个对象到底是什么。 于是我们可以在该对象中加入对请求的安全检查，包过滤，信息流追踪等，调试器可以依赖中介透明化地插入调用过程中，甚至可以由此实现懒加载。
  能力还支持安全有效的权限授权，Alice 可以为自己的对象创建一个能力，交给 Bob，然后 Bob 就可以使用该能力来操作对象，Alice 也可以使用中介技术监视该过程。

Faasm轻量级隔离实现高效的有状态无服务器计算

云计算中的无服务器计算正在成为部署数据密集型应用程序的一种流行方式。一个函数即服务（FaaS）模型将计算分解为多个函数，可以有效地利用云的海量并行性。
现有的无服务器平台将功能隔离在短暂的、无状态的容器中，防止它们直接共享内存，迫使用户反复复制和重复序列化数据，增加了不必要的性能和资源成本。
Faasm 提出了一种新的轻量级隔离方法，它支持函数之间直接共享内存，减少资源开销。（与Redleaf的解决方案有异曲同工之妙）

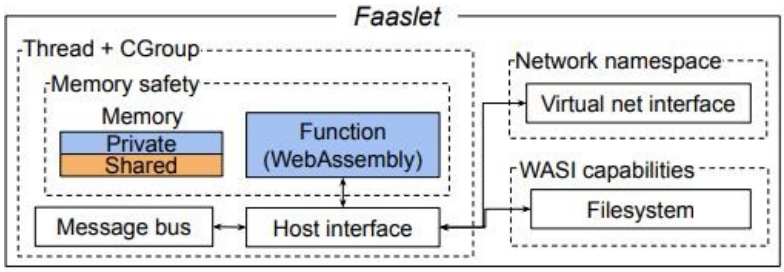


Figure 1: Faaslet abstraction with isolation

默认情况下，一个函数被放置在它自己的私有连续的内存区域中（Faaslet也支持内存的共享区域），这使得Faaslet能在WebAssembly的内存限制范围内访问共享的内存。

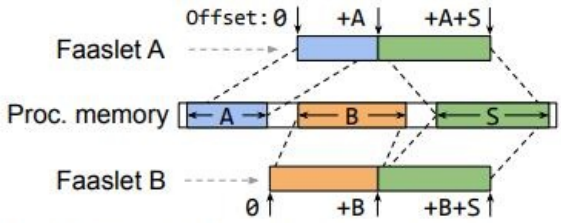


Figure 2: Faaslet shared memory region mapping

为了创建一个新的共享区域，Faaslet扩展了函数的线性字节数组，并将新的页面重新映射到进程内存的指定区域。
该函数访问线性内存的新区域因此保持了内存安全，但是基础内存访问被映射到共享的区域。
FAASM执行在Faaslets内的函数，Faaslets提供内存安全和资源公平性，并可以共享内存中的状态。

用轻量级协同内核实现性能隔离

性能隔离正在成为高性能计算（HPC）应用的一个要求，特别是当HPC架构转向原地数据处理和应用组合技术以提高系统吞吐量时。
这篇论文提出了Pisces，一个系统软件架构，使多个独立的、完全隔离的OS/Rs(enclaves)共存，它们可以被定制以解决下一代HPC工作负载的不同要求。每个飞地由专门的轻量级操作系统联合内核和运行时间组成，能够独立管理动态分配的硬件资源的分区。

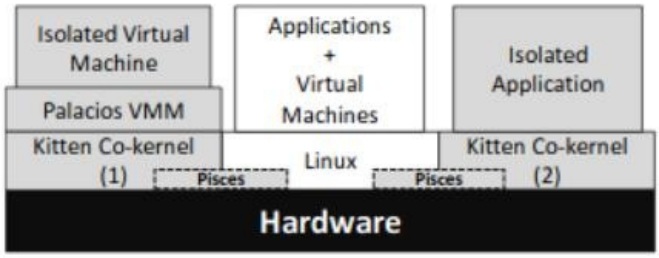
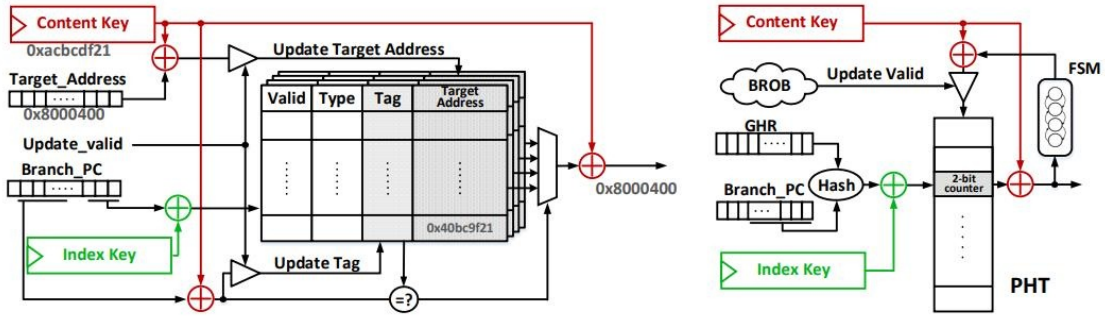


Figure 1: **The Pisces Co-Kernel Architecture**

安全分支预测器的轻量级隔离机制

不同进程共享的分支预测器给攻击者留下了恶意训练和感知的机会。作者希望通过一些使用随机化的轻量级处理来实现对这些硬件表中内容的隔离，而不是基于冲洗覆盖或物理隔离的硬件资源。(XOR)逻辑隔离的目的是防止对物理上共享的分支预测器的攻击，只允许其所有者访问。这可以采取的形式是在每个条目上添加某种线程ID到每个条目。
作者用线程私钥来转换索引和表的内容，使用 xor 操作作为编码/解码（XOR-BTB）。
索引和表的内容都用线程私有密钥进行转换；这些密钥在某些条件下会发生变化。



(a) Noisy-XOR-BTB　　(b) Noisy-XOR-PHT

现有的几种隔离方式

虚拟机

虚拟机隔离机制是通过虚拟化技术虚拟出资源完全独立的的主机，是在硬件和操作系统以上的。虚拟机有特有的 Hypervisor 层，Hypervisor 层为虚拟机提供了虚拟的运行平台，管理虚拟机的操作系统运行。每个虚拟机都有自己的系统和系统库以及应用。虚拟机能为应用提供一个更加隔离的环境，不会因为应用程序的漏洞给宿主机造成任何威胁。在各种隔离方式中，虚拟机是隔离效果最好的。

容器

容器技术与传统的虚拟化技术不同，容器直接运行在操作系统内核之上的用户空间，因而更加灵活和便捷。虚拟机的硬盘使用量一般都在 GB 量级，而容器一般在 MB 量级。容器没有 Hypervisor 这一层，并且每个容器是和宿主机共享硬件资源及操作系统，因此性能损耗相对较小。传统虚拟化技术是对硬件资源的虚拟，容器技术则是对进程的虚拟，从而可提供更轻量级的虚拟化，实现进程和资源的隔离。容器之间的进程也是相互隔离的，其中一个容器的变化不会影响其他容器。

Unikernel

Unikernel是精简专属的库操作系统(LibraryOS)，是一个专门的、单一地址空间的操作系统。
Unikernel 是与某种语言紧密相关的，一种 Unikernel 只能用一种语言写程序。它将操作系统的最少必要组件和程序本身打包在一起，直接运行在虚拟层或硬件上，显著提升了隔离性和效率。Unikernel 是技术发展的必然产物，具有小（几兆甚至更小）、简单（没有冗余的程序，没有多进程切换）、安全（没有不必要的程序，使其受到漏洞攻击的可能大大降低）、高效（运行速度相当快）的特性

进程隔离

进程隔离是为了避免进程 A 写入进程 B 的情况发生，其对硬件有一些基本的要求，其中最主要的硬件是 MMU（Memory Management Unit，内存管理单元）。进程隔离的安全性通过禁止进程间内存的访问可以方便实现。在一个有进程隔离的操作系统中，进程之间允许受限的进程间交互，即为进程间通信（IPC，Inter-Process Communication）。

前瞻性分析

我们预期实现的微内核架构，如同 seL4，同样有成为一个优秀的 1 类 Hypervisor 的潜质，但同时由于该设计在运行时更接近宏内核的架构，有助于在稳定的前提下提高虚拟化的性能。
此外，我们认为 Redleaf 的内核模块有希望能够形成生态，在模块签名等安全措施得到充分完善后，我们可以根据自身需求，高度自定义地实现一个高性能的 OS，而不必担心内核内核稳定性与模块内可能存在的 Bug，这将会大幅缩短开发周期，也让硬件性能能够被发挥到极致。
最后，该设计也与 Unikernel 的设计理念高度相似，二者有望共同取得发展。

参考文献

[1] TencentOS Tiny (https://github.com/OpenAtomFoundation/TencentOS-tiny)
[2] Huawei LiteOS (https://github.com/LiteOS/LiteOS)
[3] Operating Systems for Low-End Devices in the Internet of Things: a Survey (https://hal.inria.fr/hal-01245551/document)
[4] 树莓派 Linux 操作系统大全 (https://zhuanlan.zhihu.com/p/105299943)
[5] 树莓派3B快速上手 OpenHarmony (https://gitee.com/xfan1024/oh-rpi3b)
[6] 手把手带你做LiteOS的树莓派移植 (https://zhuanlan.zhihu.com/p/421025403)
[7] Flexos: Making os isolation flexible (https://dl.acm.org/doi/pdf/10.1145/3458336.3465292)
[8] The seL4 Microkernel An Introduction (https://dl.acm.org/doi/pdf/10.1145/3458336.3465292)
[10] Faasm 轻量级隔离实现高效的有状态无服务器计算 (http://www.prognosticlab.org/~jarusl/pubs/hpdc-2015-pisces.pdf)
[11] 带有虚拟化和虚拟的轻量级内核隔离虚拟机功能 (https://www.cse.psu.edu/~trj1/papers/vee20.pdf)
[12] 用轻量级协同内核实现性能隔离 (https://www.usenix.org/system/files/atc20-shillaker.pdf)
[13] 安全分支预测器的轻量级隔离机制 (https://arxiv.org/pdf/2005.08183.pdf)
[14] RedLeaf: Isolation and Communication in a Safe Operating System (https://www.usenix.org/conference/osdi20/presentation/narayanan-vikram)
[15] Understanding the Overheads of Hardware and Language-Based IPC Mechanisms (https://dl.acm.org/doi/pdf/10.1145/3477113.3487275)
[16] SkyBridge: Fast and Secure Inter-Process Communication for Microkernels (https://www.cse.unsw.edu.au/~cs9242/19/exam/paper1.pdf)