seL4: Formal Verification OS Kernel

阅读报告

江梦　P14206009

本文提出了一种经过形式化验证证明功能正确性的通用操作系统内核——seL4，它是一个基于 L4 的第三代操作系统内核。L4 是一组基于微内核构架的操作系统内核，澳大利亚研究组织 NICTA 创造了一个新的 L4 版本，称为 Secure Embedded L4（简写 seL4），宣布在世界上率先开发出第一个正规机器检测证明 （formal machine-checked proof）通用操作系统。本文即该组织发表的关于 seL4 的论文，文章介绍了 seL4 的架构体系和设计特点，以及对 seL4 内核从抽象规约层到 C 语言实现层的形式化机器验证。此外，本文还提出了一套融合了传统操作系统研发技术和形式化方法技术，用来快速实现内核设计与实现的方法，经过实践证明，利用这套方法学开发出的操作系统不仅充分保障了安全性，并且也不会使性能降低。

操作系统内核的可靠性和安全性几乎与计算机系统等价，因为内核作为系统的一部分在处理器的最高权限上工作，可以随意的访问硬件。因此，操作系统内核实现中出现的任何一个微小的错误都会导致整个计算机系统的崩溃。通常认为任何规模的代码中，错误都是无法避免的。因此，当安全性或可靠性尤为重要时，传统的一些做法是减少高权限的代码的数量，从而避免 bug 出现在较高的权限层内。那么，如果代码的数量较少，便可以通过形式化的机器验证方法来证明内核的实现满足规约，并且在实现时不会出现由于程序员编码而引入的实现漏洞。

本文提出了 seL4——L4 微内核家族的一员，它通过机器辅助和机器检验的形式化证明高度保证了功能正确性。目前，seL4 是第一个经过完全形式化验证功能正确性的操作系统内核，所以，它是一个前所未有的具有高度安全性和可靠性的底层系统级平台。本文中所证明的 seL4 的功能正确性要比模型检验、静态分析以及采用类型安全编程语言实现的内核要更强更精确。具体来说，seL4 的优势有以下几点：（1）适用于现实生活，并且其性能能与当前最好的微内核相媲美；（2）其行为在抽象层进行了精确的形式化规约；（3）可以形式化证明它的实现满足规约（4）形式化证明了它的访问控制机制能提供高强度的安全性保证。

首先，作者对 seL4 的设计过程进行了简要概述。seL4 的主要特征为虚拟地址空间、线程、进程间通信以及与多数 L4 内核不同的用于授权的功能。seL4 最初的开发和证明工作都

是在一个基于 ARMv6 的平台上实现的，后来添加了 x86 的端口。由于对硬件资源的高效管理和使用可以带来直接的性能提升，多数操作系统开发者倾向于自底向上的内核开发方法。与此不同的是，形式化方法的研究者们更倾向采用一种自顶向下的设计方法，这种设计的基础是一个对硬件高度抽象的简单模型。本文混合折中了上述两种方法，采用一种基于一个易于被 OS 开发人员和形式化方法从业人员访问的中间目标的方法，它使用函数式编程语言 Haskell 为 OS 开发人员提供一种编程语言，同时提供一种可以自动翻译并应用于定理证明工具的人工产品。图 1 详细描述了这种方法，图中的大矩形包括了形式化验证使用的所有模块，双箭头表示实现或证明，单箭头表示一个模块对另一个模块的设计/实现影响，中央的模块是内核的 Haskell 原型。Haskell 原型需要设计和实现管理底层硬件细节的算法，为了在与现实相近的环境中运行此原型，作者使用了由 QEMU 导出的软件来模拟硬件平台。因此，所有的仿真操作能够完整的表现出来操作系统的各种行为，同时作为 Haskell 的子集生成的形式化执行规约可以被自动翻译为定理证明工具的语言。虽然 Haskell 原型是一个可执行的接近于最终实现的模型，但它并不是最终生成的内核，还需要人工的用 C 语言重新实现内核，原因有以下几点：首先，Haskell 的运行环境包含了大量的难以验证正确性的代码；其次，Haskell 的代码依赖于内存回收机制，并不适用于实时环境；最后，使用 C 语言可以对底层实现性能进行优化。虽然可以直接将 Haskell 编译为 C 代码，但这样会减少优化内核的机会，使系统性能受到影响。

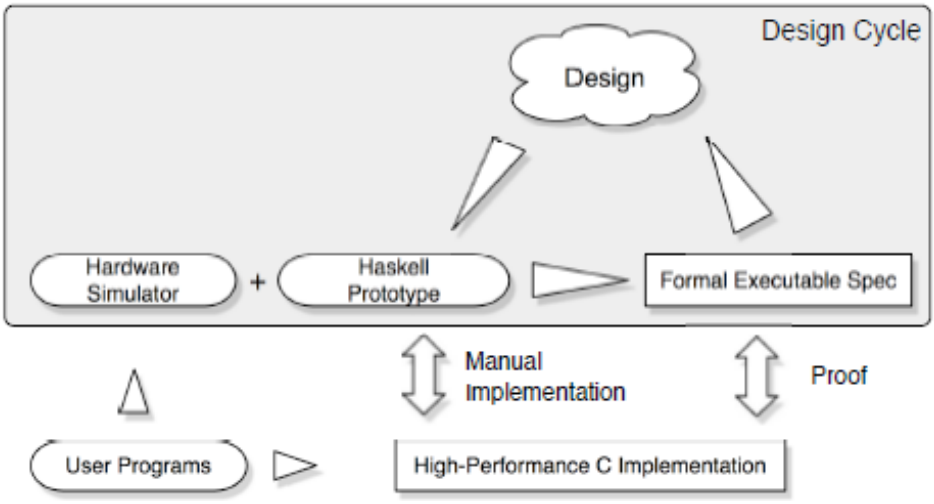

图 1：seL4 的设计过程

正确性证明的主体可以看作是每个规约层中程序语句和函数的 Hoare 元组，沿着函数边界将证明分解，每个证明单元都有一组保证先前执行的前置条件和一个含有一条或多条语句的函数来修改系统状态，以及一组后置条件。为了实现适合实际应用的内核，在设计这些组

件时应减少复杂性以便能灵活验证内核，同时不降低性能。在这个前提下，作者分析了几种内核的典型性质以及它们对证明的影响，并展示了 seL4 内核的具体设计。OS 内核普遍有包含全局变量和副作用的程序，全局变量通常需要声明和证明不变属性，这些不变量是代价高昂的，因为不仅要在本地函数还要在整个内核中证明它不会被破坏，此外若不变量暂时违规会使对全局变量的处理变得困难。为了解决这些问题，seL4 通过限制抢先节点并由 Haskell 生成代码，来明确副作用并引起设计人员的注意。在内存管理上，seL4 采用了一个内存分配模型对授权的应用进行内存分配控制，在精确保证了内存消耗的同时也有助于验证。分配算法的正确性要求保证新对象完全包含在一个未定义的内存区域中并且不会与其他对象的内存区域重叠，为此 seL4 使用一个权能派生树型结构来追踪这些内存区域。此外，在重用一个内存区域之前，其所有引用都要置无效，seL4 采用了两种方法——寻找对象的所有外部权能以及在最后的权能被删除时将对象返回给内存。seL4 通过将设备驱动程序转移到受保护的用户模式下执行，免去了 I/O 大部分的复杂性，seL4 内核中只有一个设备驱动程序——计时器驱动程序，用来执行时间片抢占。然而，seL4 仍需要处理中断。处理一个中断事件时，中断传送机制判断中断源并屏蔽其后续中断，通知用户级处理程序，并在中断被确认接收后取消屏蔽。因此 seL4 的模型中包括了控制器、中断的屏蔽以及只有非屏蔽时才能出现中断，这足以对中断控制访问等基本行为进行证明。内核可以验证这一要求使开发人员试图用最简单干净的方式实现目标，从而导致了更好的设计并减少了错误的可能性。

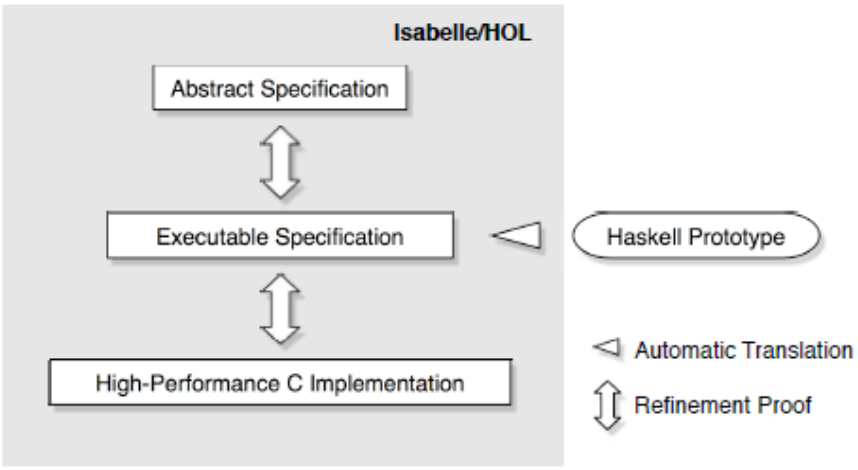

图 2：seL4 验证的规约层次

文章中所采用的形式化验证技术是交互式的机器辅助和机器检验的证明，使用的定理证明工具是 Isabelle/HOL。交互式定理证明需要人的参与，由人来创建和引导整个定理的证明，与静态分析和模型检验等自动化方法不同，这种交互式方法的优点是不受限于特定的属性或者有穷的状态空间。文中论证的是最强意义上的功能正确性，作者采取一种精炼证明，验证

了系统高层抽象和底层表示的一致性，确保了抽象层的所有 Hoare 逻辑属性在底层细化中依然适用。如图 2 所示是用于 seL4 验证的规约层次，最顶层是抽象规约层，定义了系统做些什么而没有说如何做，对于所有用户可见的操作它给出了系统预期的功能行为。在抽象层使用有限的机器字精确地描述了参数格式、编码和错误报告。此外，抽象层中使用的数据结构式高层的，如集、列表、树、功能和记录。若一个操作有很多正确结果，那么抽象层会将它们都返回给下层以便下层自由选择，这种非确定性给底层实现留下了空间。下一层是由 Haskell 原型生成的用于定理证明的执行规约层，其目的是补充抽象层遗留的细节并明确内核是如何运行的，包含了所有 C 代码实现中需要的数据结构和实现细节。为了避免 C 语言中出现如何组织数据结构和对代码优化的混乱，执行规约层对这些数据和代码结构进行了基本限制。Haskell 只是作为一个中间原型，而作者寻求的是最终 C 代码的正确性的保证，因此这个翻译过程不是严格正确的。执行规约是确定的，只留下底层机器是非确定的，在这一层所有的数据结构都是明确定义的数据类型，向前的记录或列表，能够在 C 语言中有效实现。最底层是 seL4 的高性能 C 语言实现层，从 C 到 Isabelle 的翻译过程是严格正确的，因此要求 C 语言集的语义要精确的且基本的。精确意味着将 C 语言的语义、类型和内存模型作为标准规定；基本意味着不仅要在高层将 C 语言行为公理化，也要尽可能的从第一个原则导出它。本文项目的一大突出成果就是形成了大量 C 语言的准确的形式化语义。验证不可能是完备的，因此仍需要进行适当的假设，本文中证明工作止于源代码层，即假定编译器和硬件是正确的。

本文的相关研究工作有很多，其中比较有代表性和借鉴意义的如下。UCLA Secure Unix 和 Provably Secure Operating System (PSOS)在十九世纪七十年代末第一次尝试着来验证操作系统内核。本文借鉴了 UCLA 的功能正确性的验证思路。UCLA 项目完成了 90%的规约和 20%的验证，最终得出结论——不变式的推理占据了证明的大部分时间，在本文的项目中这一点也得到了证实。PSOS 主要关注于内核设计的形式化验证，然而，却从来没有完成过大量的实现证明。他们的设计方法学被 Ford Aerospace 的 Kernelized Secure Operating System (KSOS) 、Secure Ada Target(SAT)以及 Logical Coprocessor Kernel (LOCK)所采用。第一个实际的完整证明是由 KIT 实现的，然而它针对的对象是一个高度理想化的操作系统内核，并不能在实际的机器上运行。其他的一些关于操作系统内核的形式化建模和证明都没有在实现层进行验证，包括 EROS 内核，基于 FLASK 的 SELinux 内核等。VFiasco 项目和气候的 Robin 项目尝试验证 C++的内核实现，他们创建了大量的基于 C++实际代码的模型，然而在形式化验证方面并没有做太多的工作。Heitmeyer 等人声称验证和标准化了一个名为 LOC 的嵌入式操作系统，然而，实际上他们的工作并未达到 C 代码层面的验证，与功能性验证差距较远。

比较新的一个项目是 Verisoft，他们尝试验证操作系统的内核和从硬件到应用程序的完整的软件栈。这里面包含了类似于 Pascal 语言的已经验证过的编译器，这个编译器并不做任何优化处理。虽然这个项目目前还没有完成，但是，却证实了功能性验证软件栈的目标是可以实现的。同时，他们也证明了形式化验证汇编一级的代码是可以实现的。不幸的是，他们的工作是基于 VAMP 硬件平台，该硬件平台并不被广泛使用。本文提到的 seL4 操作系统针对的是 ARM6 平台，目的是实现一个可用的真实的内核。其他用于提高操作系统可靠性的形式化技术包括静态分析、模型检验和形态分析。静态分析在最好的情况下仅仅能检测出类中内存的泄露。模型检验能够验证实际的 C 代码中的某些安全特性比如系统调用等。然而，这些自动验证技术远远不能满足功能正确性的需求。

总之，本文形式化的验证了 seL4 操作系统内核，证明了完整的、严格的、形式化的验证一个通用操作系统内核是完全可以实现的。虽然文中并没有对内核的优化投入大量精力，但也侧面展示了优化是可行的而且这种验证并不需要牺牲性能。seL4 内核可以应用于实际，运行在 ARM6 或者 x86 平台架构上。此外，对内核验证的另一好处是形成了一种可以快速设计内核的方法原型。作者通过观察形式化方法和操作系统方面两类设计原则的融合，得到了一些设计决策，如基于事件的内核主要是非抢占式的并使用中断轮询，这些决策使内核设计更简单并更易于验证。此项目的下一步工作可能包括对汇编代码部分的证明，多核的内核，以及应用程序的验证。我认为其中对应用程序的验证更容易实现且更有意义，因为应用程序的证明可以在 seL4 已经证明实现的抽象的形式化内核规约基础上进行，并且这也是对 seL4 应用的一种扩展。此外，本文主要对 seL4 的代码级的功能正确性进行了证明，我认为接下来可以更进一步验证硬件上执行的二进制代码是 C 代码的正确转化。