第 38 卷 第 11 期
Vol.38 No.11

计 算 机 工 程
Computer Engineering

2012 年 6 月
June 2012

·开发研究与设计技术· 文章编号：1000—3428(2012)11—0234—05 文献标识码：A 中图分类号：TP316

操作系统形式化设计与验证综述

钱振江[a,b]，刘 苇[a,b]，黄 皓[a,b]

(南京大学 a. 软件新技术国家重点实验室；b. 计算机科学与技术系，南京 210046)

摘 要：对操作系统的形式化设计和验证的概念进行介绍，描述其框架和基本方法。比较和分析操作系统宏内核和微内核结构，调查多个设计和验证项目，阐述项目的验证目标、方法、优缺点和进展情况。在总结研究现状的基础上，分析和展望操作系统形式化设计和验证的发展趋势，从操作系统模型设计、验证工具、代码实现和验证重用等方面给出形式化设计和验证的思路。

关键词：操作系统；形式化设计；形式化验证；定理证明；系统安全；框架设计

Survey of Formal Design and Verification for Operating System

QIAN Zhen-jiang[a,b], LIU Wei[a,b], HUANG Hao[a,b]

(a. State Key Laboratory for Novel Software Technology; b. Department of Computer Science and Technology, Nanjing University, Nanjing 210046, China)

【Abstract】 This paper introduces the concepts of formal design and verification for the Operating System(OS), and elaborates the framework and foundational methods of formal design and verification. It compares and analyzes the monolithic kernel and micro kernel architectures of the OS. Meanwhile, it investigates multiple design and verification projects in depth, focusing on the verification objectives, the methodology, the advantages and limitations, and the progression. It summarizes the state of the art, analyzes and outlooks the trends of the formal design and verification for the OS. What is more, from the aspects of model design, verification tools, code implementation and verification reuse, it advances the ideas of formal design and verification.

【Key words】 Operating System(OS); formal design; formal verification; theorem proving; system security; architecture design

DOI: 10.3969/j.issn.1000-3428.2012.11.072

1 概述

计算机系统的信息安全是当前信息技术领域极其重要的研究方向。信息安全技术的进展推动了信息化进程，但是在许多领域也在阻碍着信息化的推进。学术界和产业界已经进行了艰苦卓越的工作，但是信息系统的安全仍然充满了不确定性，没有任何一个安全产品可以保证完全地实现理想的安全目标。最主要的原因在于计算机系统的核心——操作系统(Operating System, OS)往往无法提供理想的安全服务和安全保障。这主要存在 2 个方面的原因：(1)在 OS 实现过程中，OS 开发者不可避免地存在编程错误、实现与设计不一致等问题；(2)更重要的是，在 OS 设计过程存在 OS 功能设计与安全目标不一致等问题。为此，国内外的很多学者对 OS 的设计方法进行了多方面的研究。

目前，公认的、最为合理的方案是利用严格的形式化方法来对 OS 进行设计实现和验证。形式化方法是指使用基于数学逻辑和各种推理验证的技术来描述、开发以及验证软件和硬件系统的方法。利用形式化方法对 OS 进行设计和验证，包括利用严格的数学形式方法描述 OS 的操作行为和语义，建立 OS 语义模型，并验证其代码实现了设计预定的功能需求。这种方法称为形式化验证(formal verification)。形式化验证包括模型检测(model checking)[1]和定理证明(theorem proving)2 种方式。模型检测方式主要是利用对系统问题建立的数学模型进行自动推理；定理证明一般采用交互式的定理辅助证明器来对系统问题进行抽象描述，并以数学公式定理的方式表达系统的功能和安全性，采用数学定理推导演算的方法进行验证。值得一提的是，就目前而言，定理证明方式可以验证的问题数量显然比模型检测来得多。本文从定理证明的角度，就近年来操作系统形式化设计与验证问题的发展、思想，以及存在的一些问题进行了总结。

2 OS 形式化设计和验证框架

OS 的形式化设计和验证的框架如图 1 所示。

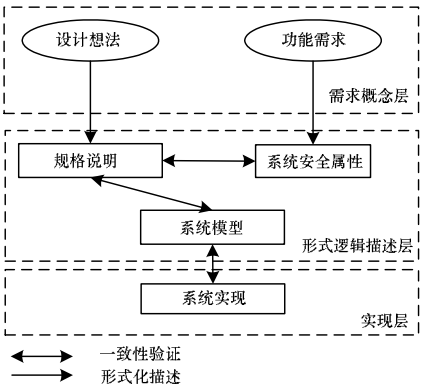

图 1 OS 形式化设计和验证框架

基金项目：国家“863”计划基金资助项目“分布式可信计算系统研究”(2007AA01Z409)；国家自然科学基金资助项目(60721002)；江苏省科技支撑计划基金资助项目“可信操作系统新技术研发”(BE2008124)

作者简介：钱振江(1982－)，男，讲师、博士研究生，主研方向：操作系统安全，形式化验证，嵌入式系统；刘 苇，硕士研究生；黄 皓，教授、博士、博士生导师

收稿日期：2011-10-15 **E-mail：**zhenjiang.qian@gmail.com

图 1 中的顶层是存在于 OS 设计者头脑中的概念和想法，如 OS 的功能需求、基本工作原理、OS 应满足的安全属性和策略等。这些都只是存在于设计者头脑中的内容，并不是形式化的逻辑表达，不能用来进行推理证明。

中间层是形式逻辑描述层，包括系统的规格说明、安全属性描述和系统形式化模型。需求概念层中非形式化的概念可以转化为 OS 规格说明中的属性和公式项，如非形式化命题“进程空间保持隔离性”、“用户 U 必须得到相应的权限才能访问资源 R”转化为公式项可以表示为“$\forall p_1 \in Process, p_2 \in Process, Space(p_1) \wedge Space(p_2) = \varphi$”和“$HasCapability(U, R) \rightarrow Access(U, R)$”。同时，规格说明还包括对 OS 的行为以及系统状态改变的抽象描述。在 OS 的规格说明的基础上构建 OS 的形式化模型，该模型对系统中所有的主体(如进程、内核等)以及资源(如内存、运行时间)抽象成对象，称为 OS 对象，并采用一定的逻辑系统(如一阶逻辑、二阶逻辑以及高级逻辑[2]等)，将 OS 对象、OS 的行为、状态转化以及安全属性等使用形式语义(如操作语义、指称语义以及公理语义[3]等)进行描述，从而获得 OS 语义模型。这一中间层也是后期 OS 验证的基础，用于进行 OS 一致性验证，包括 OS 功能和安全目标与 OS 形式化设计的一致性、OS 形式化设计与 OS 代码实现的一致性。

最底层是系统实现层，包括系统的软硬件实现和系统运行所处的外部环境等因素。对这一层的验证目前主要是进行系统模型和代码实现的一致性验证。由于硬件和系统运行环境涉及非形式化的因素，很难用形式化的方法来描述，为此对于该层的完整形式化描述和验证尚无完善的解决方案。目前，很多的研究者都根据验证的需要对系统进行不同层次的抽象，如将系统抽象成高级语言层、汇编层以及机器码层，相应的验证工作包括各层的正确性以及层与层之间的一致性。

3 微内核结构

本节介绍 OS 的微内核结构，这也是重点描述的 OS 验证的对象结构。

内核是 OS 的核心部分，从定义上来说，内核是运行在硬件特权模式的软件系统。这种特权模式相对于普通用户模式而言，主要是具有访问硬件寄存器和设备的特殊机器指令，这些硬件寄存器和设备在用户模式是不可见或者只读的，如 CPU 处理器的内存管理单元(Memory Management Unit, MMU)。

从 OS 设计原则上来说，与传统的宏内核结构相比，微内核结构将运行在特权模式的系统代码量降至尽可能得少。宏内核和微内核 OS 结构如图 2、图 3 所示。

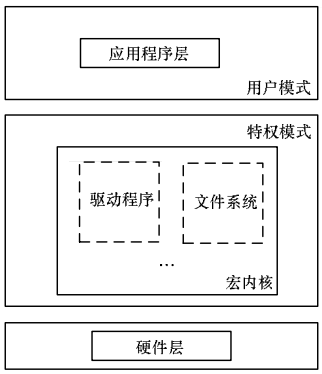

图 2 宏内核 OS 结构

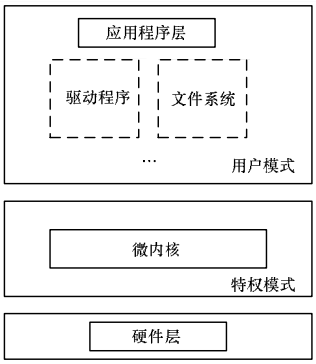

图 3 微内核 OS 结构

宏内核 OS(如 Windows、Linux 等)将驱动程序和系统服务(如文件系统等)放入内核中，一起运行在特权模式。微内核 OS(如 Minix[4]、Mach[5]、L4[6]、VAMOS[7]等)中只有微内核自身运行在特权模式，而驱动程序、系统服务和应用程序都运作在用户模式。

在宏内核中，任何的驱动程序都有权访问所有的内存和内核数据结构，为此，驱动程序中的任何一个错误都有可能使整个系统崩溃。文献[8-9]说明了这种错误的可能性。对于微内核结构，这种可能性很小，因为驱动程序都运行在自身的保护域中，驱动程序中的错误只会影响到自身所用内存的数据，不会对整个系统造成危害。

微内核结构也存在自身的问题，其中很多学者最为关注的是性能问题。由于微内核中的执行主体往往采用消息进行通信，这种通信方式非常耗时，从而影响了整个系统服务的性能[10]。但是，目前经过很多学者的研究，可以将微内核消息通信的延时降低到和一个 C 语言的过程调用相同的数量级，这样一来，微内核完全可以发挥出理想的性能，如文献[6]实现的 L4 微内核 OS 在消息通信上表现突出，文献[11]介绍了 L4 上高性能用户层网络设备驱动的实现。

4 OS 设计和验证研究现状

本节将对国内外在 OS 设计和验证方面有代表性的项目进行阐述，这些项目都对 OS 内核或者 OS 整体的安全性和正确性进行了一定程度的设计和验证。由于篇幅限制，主要对微内核架构的项目进行分析和比较。

在 OS 安全验证方面，SELinux[12]是典型的代表。在国内，许多的研究机构从 20 世纪 90 年代开始也进行过大量的工作，包括麒麟、红旗 Linux、安胜操作系统[13-14]等。

4.1 UCLA Secure Unix

在形式化工作方面，OS 的形式化验证工作可以追溯到 20 世纪 70 年代。1978 年，UCLA 为 PDP-11 机器开发了 UCLA Secure Unix[15]，给出了 Unix 用户界面的安全内核原型，证明了部分的抽象层次规约的一致性，但是没有证明所有内核层次之间以及与实现的一致性。UCLA 设计和验证结构如图 4 所示。

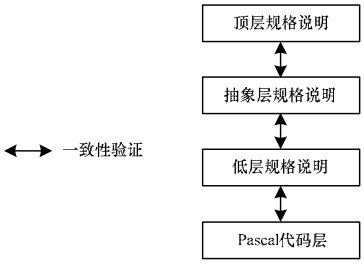

图 4 UCLA Secure Unix 规格说明层结构

图 4 说明了 UCLA Secure Unix 项目验证过程中用到的各种规格说明层。最低层是内核的 Pascal 代码，包含所有实现的细节。最上层是顶层规格说明，描述了内核的权限访问控制模型。中间部分是抽象层规格说明和低层规格说明，这 2 层都使用相同的形式化描述，它们的区别只是抽象的层次不同，低层规格说明包含了在内核调用接口中要用到的所有变量对象；抽象层规格说明主要是一些抽象的数据结构，如列表和集合，并且包含所有在该层可见的内核操作。这些规格说明都采用了状态机的方式来描述，包括所有可能的状态、当前状态以及状态之间可能的转化操作。系统的验证环节主要是阐明这些规格说明是一致的，也就是说状态机能相互之间进行描述。

图 5 表示了 UCLA Secure Unix 中规格说明层之间一致性验证方式。通过定义状态映射函数 f：real_state→abstract_state，将系统实现层的每个状态映射到抽象层的对应状态。在具体的验证过程中，通过说明系统实现层中每个操作引起的状态转化在抽象层都有相应的抽象操作引起的状态转化来表达层次间的一致性。

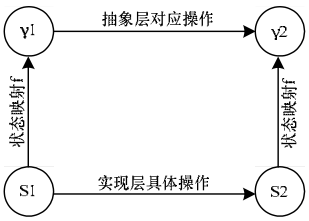

图 5 UCLA Secure Unix 规格说明层一致性验证方式

4.2 PSOS

Provably Secure Operating System(PSOS)是由 SRI 国际组织[16]于 1973 年-1980 年设计的，其目的在于提供一个可证明安全性的通用 OS。PSOS 提出了多级分层抽象的方法，对硬件和软件架构的抽象描述是类型安全(type-safety)的。其采用的规约和断言语言 SPECIAL[17]用于精确定义各个层次的模块以及层与层之间的抽象映射。PSOS 只提供了一些简单的实例来说明实现和规约是一致的，没有做到真正的形式化设计和验证。不过，PSOS 提出了很多 OS 验证和系统设计的重要理论和方法，其系统分层架构如图 6 所示。

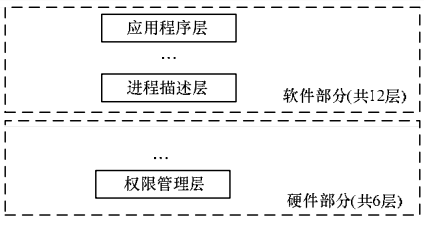

图 6 PSOS 分层系统结构

PSOS 的设计将系统分成 18 层，其中，底下的 6 层计划使用硬件来实现。PSOS 并没有内核的概念，但设计过程充分运用了硬件抽象、模块化封装和信息隐藏的技术。值得一提的是，最低层的权限管理层作为整个系统的访问控制功能实现对系统资源的访问管理，该层是通过硬件来实现的，硬件使用一个额外的比特位来识别内存中的一个字是否存储了权限信息。这些内存字需要特殊的硬件权限指令来访问，并且这些权限信息是不能通过指令来修改和擦除的。

4.3 DTOS

1992 年，美国支持完成的 LOCK 项目致力于开发达到或超过 A1 级的安全 OS，其整个开发周期的 58%的工作都被用来进行形式化安全策略模型和顶层规范的设计，以及策略与设计之间的一致性证明。在 1992 年-1993 年间，美国国家安全局(NSA)和安全计算公司(SCC)的研究人员在 LOCK 项目的基础上，共同设计和实现了分布式可信 Mach 系统(DTMach)。DTMach 项目的后继项目——分布式可信 OS(DTOS)[18]最终形成了 Z 语言版的形式化安全策略模型(FSPM)和形式化顶层规范(FTLS)技术文档，研究了形式化方法的应用，但未见对整个 DTOS 进行验证的报告。

4.4 KIT

Kernel for Isolated Tasks(KIT)是一个小型的操作系统内核，由德克萨斯州大学奥斯汀分校实现。Bevier 在其博士论文[19]中描述了 KIT 的形式化验证。KIT 内核主要提供了进程调度、错误处理、消息传递、字符 I/O 设备驱动等。但是，KIT 缺乏进程动态创建以及共享内存和按需分页的功能。KIT 在上层设计的过程中保证“一个进程的执行不能以非法的形式干涉其他进程的运行”。KIT 虽然只是作为特殊用途的操作系统的内核使用，但是 KIT 项目是第一个采用机器化方式自动进行形式化验证的例子。

4.5 EROS

1995 年，Charlie Landau 组织了 EROS[20]项目，它的形式化验证工作主要考虑地址转换的正确性和内核的内存使用的安全性部分。EROS 的后继项目 Coyotos 由 Hopkins 在 2006 年组织，开发了一个通用 OS，并设计一种低层次的编程语言 BitC，同时给出相应的形式语义[21]，阐述了 Coyotos 架构和形式化验证的方法[22]。Coyotos 的形式化验证工作的还没有进一步结果。

4.6 VFiasco

2002 年，文献[23]报告了 VFiasco 项目，它由德累斯顿技术大学组织对兼容 L4 的 OS 微内核 Fiasco 进行形式化验证，采用了 SPIN 模型检测的方法来验证 IPC 机制，同时采用定理辅助证明工具 PVS[24]进行建模和代码验证，对 C++语言进行了部分改进加强编程语言的安全性。但是 VFiasco 没有对系统的设计进行验证，所以，实际的设计结果存在缺陷。2008 年，文献[25]报告了 VFiasco 的工作在 Nova Hypervisor 项目中的延续，提出了 IA32 处理器的模型和内存模型，实现了将 C++代码直接转换为相应 PVS 语意代码的工具。

4.7 Robin

Robin[26]项目开始于 2006 年，该项目旨在开发一个具有最小的可信计算基的虚拟机监控器(Micro Hypervisor)，能够安全地虚拟出多个传统 OS 的实例[27]，Robin 项目使用 PVS 进行了形式化描述，验证了一个简单版本的 x86 硬件模型以及一个 C++的子集语义的正确性，迄今未见 Robin 完整的技术报告。

4.8 L4.verified

L4.verified 项目由澳大利亚国家 ICT 实验室(NICTA)在 2004 年-2006 年实施发起的。该项目重点在形式化验证 L4 微内核的 ARM 体系结构版本——L4Ka∷Pistachio[28]。其正确性证明采用 Isabelle/HOL[29]形式化证明辅助工具。文献[30]介绍了验证的工作。验证的目标是使用 Isabelle/HOL 证明最终的执行效果符合抽象模型的预期定义。然而该项目的验证只考虑到 C 语言层次，并不考虑编译器产生的汇编代码与 C 语言之间的一致性，因此，采用的编译器并没有经过形式化验证。该项目提出对于内核验证必须做到 3 个抽象层级之间

的一致性，即 C 语言的实现层、执行的规约描述层，和抽象内核模型层。L4.verified 项目还使用访问控制模型证明了系统的一些安全属性，并计划证明抽象内核模型层与访问控制模型层的一致性，从而证明 C 语言正确实现了的这些安全属性。完整的 L4 内核一共有 8 700 行 C 语言程序以及 600 行汇编程序。目前，其中的虚拟内存管理，1 200 行的初始化代码(C 语言实现)以及这 600 行的汇编代码并没有得到验证。L4.verified 的系统层次如图 7 所示。

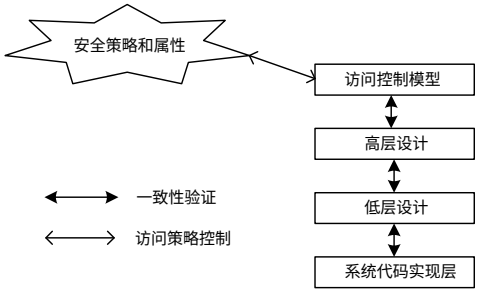

图 7 L4.verified 分层结构

L4.verified 的系统分层方式与 UCLA Secure Unix 比较类似。在最底层是使用 C 语言和汇编语言的代码实现层。为了对实现的正确性进行推理验证，在代码实现层包含了描述硬件细节的模型，如 MMU、异常和中断的模型。在代码实现层之上是低层设计层，包含了所有重要数据结构的实现，如存储权限的比特位、使用指针实现的双向链表等。在低层设计层之上是高层设计层，包含了对所有用户可见的内核操作的描述，但不涉及具体的数据结构和实现细节，如在高层设计中对于内核线程调度功能描述为“调度系统中任何可以运行的线程”，但在低层设计中需要使用一个数据结构来记录哪些线程是可以运行的以及它们的优先级。

L4.verified 的最上层是访问控制模型，描述了系统中资源的访问权限的分布情况。文献[31]对 L4.verified 的内核进行了安全策略和属性的验证，表明其内核能有效地实现子系统之间的隔离，并能提供灵活的全局安全策略。

4.9 Flint

从 20 世纪 90 年代开始到现在，Shao 等人带领的 Flint 项目组在形式化验证方面做了大量的工作[32]。在安全语言方面，Flint 开发了一种新的编程语言 VeriML[33]，其采用的逻辑系统 λHOL^{ind} 在 λHOL 逻辑[34]的基础上加入了对数据类型的归纳定义，提供了丰富的形式化描述能力以及类型安全特性。同时，Feng 等人提出的 OCAP[35]开放逻辑框架首次成功将 OS 不同模块的验证逻辑结合起来，形成一个完整的验证系统，并保证验证模型的可扩展性。目前，Flint 项目已初步完成了用 VeriML 实现 OS 内核[36]，但是还没有对 OS 进行验证。

4.10 Verisoft

Verisoft 是一个大型的计算机系统设计开发项目，目标在于对整个计算机系统自底向上从硬件层到应用软件层进行普适形式化证明(pervasive formal verification)。整个项目从 2003 年开始，直至 2007 年开发计划终止，耗时 4 年时间。在 2007 年，其后继项目 Verisoft XT[37]正式启动。在 Verisoft 项目中，普适形式化证明意味着整个项目重点关注的问题并不是编译器和机器指令模型的正确性，而是整个系统的设计层次都要经过严格的形式化验证，从而组成一个完整的软硬件的验证链。Verisoft 项目的框架如图 8 所示。

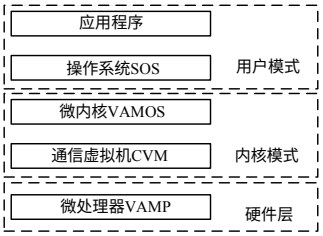

图 8 Verisoft 分层结构

Verisoft 的设计方法和传统的 OS 设计类似，采用了分层结构，最顶层为应用软件层，最底层为硬件层。其中的每个层次在设计前期都经过了手动的验证，75%左右的验证都使用 Isabelle/HOL 进行机器同步验证。

Verisoft 系统框架中处于最低层的硬件层主要包括一个微处理器 VAMP[38]，这是一个 RISC 架构的处理器，支持简单的地址转换、内存映射 I/O 设备[39-40]以及管道。Verisoft 小组利用 PVS 对 VAMP 的指令集行为进行了形式化验证。

在 VAMP 之上的是通信虚拟机 CVM(Communicating Virtual Machines)[41]，包括了 OS 内核与硬件相关的部分，为 OS 内核提供了与硬件无关的接口，是 OS 内核和硬件之间的中间层。这一层主要是为了 OS 内核验证的便利性而设计的，它将 OS 内核实现代码中与硬件相关的汇编部分(如驱动程序)分离出来单独进行验证。这一层的验证工作已基本完成[41]。

在 CVM 之上是一个微内核 VAMOS，包括运行在硬件特权模式的代码。VAMOS 和 CVM 一起组成 OS 内核。文献[42]说明了 VAMOS 的验证工作，目前仍在进行中。

在 VAMOS/CVM 内核之上是操作系统 SOS(Simple Operating System)[43]。SOS 作为特权用户进程来运行，拥有很多的对硬件资源的直接访问权限。SOS 实现了 TCP/IP 协议和文件系统，其验证工作尚未完成。

最上层的应用程序层主要是一些应用程序的例子，其中包含一个小型的邮件客户端[44]。该层的大部分软件都得到了验证。

5 结束语

本文对 OS 的形式化设计和验证的框架以及目前的发展状态进行了阐述。目前还没有一款通用 OS 或者 OS 内核可以认为是经过完全形式化验证的，或多或少地存在着非形式化的描述和验证。即使是在很大程度上经过形式化设计和验证的项目，如何将系统的实现代码自动化地转化为验证工具的验证代码仍然是困扰很多形式化学者的重要问题，这其实也是现实世界的非形式化和逻辑世界的形式化之间的一道鸿沟。

本文阐述的项目表明 OS 的形式化设计和验证仍然是困难和费时的研究工作，但是和发展初期相比已取得了很多的进步，并形成了很多的成熟技术，如自动化验证、内存和机器模型、计算机语义以及验证逻辑等。OS 形式化设计和验证技术需要在以下的方面有所突破：(1)在 OS 模型设计方面，将系统的功能需求和安全策略的非形式化描述与 OS 模型的形式化描述有机地结合，便于后期对两者一致性的验证。(2)在形式化验证工具方面，验证工具采用的验证逻辑要更加贴近程序逻辑，便于将系统实现代码转化为验证代码。(3)在 OS 实现代码方面，所采用的编程语言要加入验证逻辑的概念，使编程语言起到描述和验证的桥梁作用。(4)在验证重用方面，对系统功能模块的验证要和程序逻辑相分离，便于将已有的验证工作应用到后续的验证任务中。

参考文献

[1] Clarke E M, Grumberg O, Peled D. Model Checking[M]. Cambridge, USA: The MIT Press, 1999.

[2] Guenthner F. Handbook of Philosophical Logic[M]. Berlin, Germany: Springer, 2007.

[3] 周巢尘. 形式语义学引论[M]. 长沙: 湖南科学技术出版社, 1985.

[4] Tanenbaum A S. The Minix Project[EB/OL]. [2011-03-12]. http://www.minix3.org/.

[5] Rashid R, Julin D, Orr D, et al. Mach: A System Software Kernel[C]//Proc. of COMPCON'89. San Francisco, USA: [s. n.], 1989: 176-189.

[6] Liedtke J. On μ-kernel Construction[C]//Proc. of SOSP'95. New York, USA: ACM Press, 1995: 237-250.

[7] Dörenbächer J. Vamos Microkernel: Formal Models and Verification[EB/OL]. [2011-03-12]. http://www.cse.unsw.edu.au/~formal methods/events/svws-06/VAMOS_Microkernel.pdf.

[8] Geeknet, Inc. Root Exploit for NVIDIA Closed-source Linux Driver[EB/OL]. [2011-03-12]. http://it.slashdot.org/article.pl?sid=06/10/16/2038253.

[9] SecurityFocus. Vulnerabilities Web Site[EB/OL]. [2011-03-12]. http://www.securityfocus.com/vulnerabilities.

[10] DiBona C, Ockman S, Stone M. Open Sources: Voices from the Open Source Revolution[M]. Sebastopol, USA: O'Reilly Media, Inc., 2008.

[11] Leslie B, Chubb P, Fitzroy-Dale N, et al. User-level Device Drivers: Achieved Performance[J]. Journal of Computer Science and Technology, 2005, 20(5): 654-664.

[12] McCarty B. SELinux: NSA's Open Source Security Enhanced Linux[M]. Sebastopo, USA: O'Reilly Media, Inc., 2004.

[13] 卿斯汉, 朱继峰. 安胜安全操作系统的隐蔽通道分析[J]. 软件学报, 2004, 15(9): 1385-1392.

[14] 卿斯汉. 高安全等级安全操作系统的隐蔽通道分析[J]. 软件学报, 2004, 15(12): 1837-1849.

[15] Walker B J, Kemmerer R A, Popek G J. Specification and Verification of the UCLA Unix Security Kernel[J]. Communications of the ACM, 1980, 23(2): 118-131.

[16] SRI International. SRI Project[EB/OL]. [2011-03-12]. http://www.sri.com.

[17] Robinson L, Roubine O. Special: A Specification and Assertion Language[M]. [S. l.]: Stanford Research Institute, 1977.

[18] Secure Computing Corp. DTOS Formal Security Policy Model[EB/OL]. (1996-09-26). http://www.cs.utah.edu/flux/fluke/html/dtos/HTML/final-docs/fspm.pdf.

[19] Bevier W R. A Verified Operating System Kernel[D]. Austin, USA: University of Texas, 1987.

[20] Shapiro J S, Smith J M, Farber D J. EROS: A Fast Capability System[C]//Proc. of SOSP'99. New York, USA: ACM Press, 1999: 170-185.

[21] Shapiro J S, Sridhar S, Doerrie M S. BitC Language Specification[EB/OL]. [2011-03-12]. http://coyotos.org/docs/bitc/spec.html.

[22] The EROS Group. Coyotos Project[EB/OL]. [2011-03-12]. http://www.coyotos.org.

[23] Hohmuth M, Tews H, Stephens S G. Applying Source-code Verification to a Microkernel: the VFiasco Project[C]//Proc. of the 10th Workshop on ACM SIGOPS. New York, USA: ACM Press, 2002.

[24] Owre S, Rushby J M, Shankar N. PVS: A Prototype Verification System[C]//Proc. of CADE'92. [S. l.]: Springer, 1992.

[25] Tews H, Weber T, Völp M, et al. Nova Micro-hypervisor Verification Formal, Machine-checked Verification of One Module of the Kernel Source Code[D]. Nijmegen, Holland: Radboud University, 2008.

[26] ROBIN Consortium. Robin Project[EB/OL]. [2011-03-12]. http://robin.tudos.org.

[28] Tews H. Micro Hypervisor Verification: Possible Approaches and Relevant Properties[EB/OL]. [2011-03-12]. http://robin.tudos.org/publications/hyperveri.pdf.

[28] Liedtke J, Dannowski U, Elphinstone K, et al. The l4ka Vision[EB/OL]. [2011-03-12]. http://www.cis.upenn.edu/~ahae/papers/L4Ka.pdf.

[29] Nipkow T, Paulson L C. Isabelle/HOL: A Proof Assistant for Higher-order Logic[M]. Berlin, Germany: Springer, 2002.

[30] Tuch H, Klein G. Verifying the L4 Virtual Memory Subsystem[C]//Proc. of NICTA Formal Methods Workshop on Operating Systems Verification. Sydney, Australia: [s. n.], 2004: 73-97.

[31] Elkaduwe D, Klein G, Elphinstone K. Verified Protection Model of the SeL4 Microkernel[EB/OL]. [2011-03-12]. http://ertos.nicta.com.au/publications/papers/Elkaduwe_GE_07.pdf.

[32] Flint Group. Yale Flint Project[EB/OL]. [2011-03-12]. http://flint.cs.yale.edu/.

[33] Stampoulis A, Shao Z. VeriML: Typed Computation of Logical Terms Inside a Language with Effects[C]//Proc. of ICFP'10. New York, USA: ACM Press, 2010.

[34] Robinson A, Voronkov A. Handbook of Automated Reasoning[M]. Amsterdam, Holland: Elsevier, 1999.

[35] Feng Xinyu. An Open Framework for Certified System Software[D]. New Haven, USA: Yale University, 2007.

[36] Shao Z, Ford B. Advanced Development of Certified OS Kernels[D]. New Haven, USA: Yale University, 2010.

[37] The Verisoft XT Consortium. The Verisoft XT Project[EB/OL]. [2011-03-12]. http://www.verisoftxt.de/.

[38] Beyer S, Jacobi C, Kroening D, et al. Putting it All Together: Formal Verification of the VAMP[J]. International Journal on Software Tools for Technology Transfer, 2006, 8(4-5): 411-430.

[39] Hillebrand M A, Rieden T I, Paul W J. Dealing with I/O Devices in the Context of Pervasive System Verification[C]//Proc. of International Conference on Computer Design. Las Vegas, USA: CSREA Press, 2005: 309-316.

[40] Alkassar E, Hillebrand M, Rusev S K R, et al. Formal Device and Programming Model for a Serial Interface[C]//Proc. of VERIFY'04. Bremen, Germany: [s. n.], 2004: 4-20.

[41] Rieden T I, Tsyban A. CVM: a Verified Framework for Microkernel Programmers[C]//Proc. of SSV'08. Amsterdam, Holland: Elsevier Science Publishers, 2008: 151-168.

[42] Starostin A, Tsyban A. Correct Microkernel Primitives[C]//Proc. of SSV'08. Amsterdam, Holland: Elsevier Science Publishers, 2008: 169-185.

[43] Bogan S. Formal Specification of a Simple Operating System[D]. Saarbrücken, Germany: Saarland University, 2008.

[44] Beuster G, Henrich N, Wagner M. Real World Verification Experiences from the Verisoft Email Client[C]//Proc. of ESCoR'06. Seattle, USA: [s. n.], 2006: 112-115.

编辑 任吉慧