**安全攸关领域中嵌入式操作系统软件验证的方法、理论和工具**

撰写人：王伟 同济大学 电子与信息工程学院

**摘要：**随着嵌入式计算的日益普及和不断深入，嵌入式操作系统的规模和复杂性急剧增大，在越来越多的系统中成为主要的使能部件。在航空航天、轨道交通、武器装备、医疗设备、交通、核能、金融等安全攸关的应用领域，操作系统的失效将导致灾难性的后果，确保嵌入式操作系统的安全质量成为迫切的需求和挑战。然而，由于其复杂度高、底层细节较多的特点，嵌入式操作系统的验证一直是一大难题。虽然国际上在这个方向上的研究已经取得了很大进展，但仍然面临巨大挑战。而国内在这个方向的研究仍然较少，仅处于起步阶段。

对嵌入式操作系统软件验证的方法、理论和工具开展研究，特别是对如下问题进行深入和系统地研究：如何有效地控制嵌入式操作系统的复杂度，使得我们可以进行模块化验证？如何开发好的验证方法和理论来验证现有技术不能验证的系统软件的复杂模块，如底层硬件相关的模块、含有大量并发或输入输出的模块等？需要哪些工具支持来提高验证的效率或者验证过程本身的可信性？如何重用现有的各种理论和工具？如何从验证中反过来列系统软件的开发提供启发和指导，以支持可信系统的开发？最后，如何将以上各种问题放在一个统一的框架下解决，并完成一个完整的实用系统的验证？

本项目针对国内目前相关领域的重大需求开展研究，可望为安全攸关的应用提供一个高可信的基础计算平台，适应对高可信软件的强烈需求。

**关键字：**安全计算、可靠性、操作系统、Safety

**1 项目背景和意义**

随着国家、社会和人们日常生活对软件系统的依赖程度日益增长，复杂软件系统的正确性、安全性（包括safety和security）和可靠性对安全攸关的基础设施和日常应用变得至关重要。高可信的安全攸关软件成了保障国家安全、保持经济可持续发展、维护社会稳定和保护人们生命财产安全与个人隐私等的必要条件。

不幸的是，软件在很多时候也是我们最缺信任感的工程产品。软件产品缺少像其他产品那样随产品附带的质量保证书。国内外由于软件缺陷而导致严重的灾难、事故和损失屡见不鲜。例如，2003年5月，由于飞船的导航软件设计缺陷，俄罗斯“联盟TMAI”载人飞船返回途中偏离了预定降落地点约460公里。2004年9月14日，由于空管软件的缺陷，美国洛杉矶机场400余架飞机与机场一度失去联系，给几万名旅客的生命安全造成严重威胁。2005年11月1日，日本东京证券交易所由于软件系统出现故障而陷入全面瘫痪。2006年，我国中航信离港系统发生三次软件系统故障，造成近百个机场的登机系统瘫痪。

操作系统软件作为直接控制计算硬件系统的基础软件，它本身的缺陷更容易导致整个计算系统的失控。现实中，因为操作系统漏洞而招致的恶意攻击也屡见不鲜。2010年被用来攻击伊朗的大量工业设施，甚至包括伊朗布什尔核电厂的令人闻之色变的“超级蠕虫病毒”Stuxnet就是通过操作系统漏洞开展攻击[1]。该病毒利用Windows系统的4个“zero-day attack”，并以设备驱动程序的方式来入侵操作系统，获取系统控制权，然后攻击运行于系统之上的西门子WinCC/PCS 7 SCADA控制软件。

操作系统软件作为各种应用软件运行的基础平台，其可靠性和安全性是各类高可信应用软件的前提。我国目前对安全可靠的系统软件也有很迫切的需求。在我国大力发展的高铁、航天、核电等关乎国计民生同时又安全攸关的领域，国家自主生产的可靠、安全的系统软件是保障国家安全和保护人们生命财产的必要条件。在这些领域，欧洲很多国家早就开始使用形式化程序验证技术来提高软件系统的可信性，例如法国城际铁路的SACEM系统[2]和空客公司的航空电子设备软件[3]等。而我国在这方面则缺乏相应的技术，很多关键操作系统软件目前还需要靠进口，特别是需要进行安全标准认证的软件。

形式化程序验证是一种严格保证软件系统可靠性的重要技术。通过验证来确保软件正确性的想法，早在上个世纪60年代就由Floyd、Hoare等人提出。近年来，由于程序设计语言理论方面的突破，以及自动定理证明和程序分析能力随着计算机处理器性能的增强而增强，形式化程序验证又重新成为当前研究的热点。2003年，程序验证的创始人之一，图灵奖得主Tony Hoare提出“verifying compiler”是计算机科学领域的一个巨大挑战[4]。2007年，Hoare等人发表《软件验证倡议宣言》[5]，号召研究人员共同合作来克服软件验证这个巨大的挑战。由于操作系统软件所处的核心地位，安全可信操作系统软件的验证近年来也获得了广泛关注。

**2 国内外研究现状**

**2.1 操作系统软件的验证**

近十年来，国际上众多大学与研究机构纷纷开展操作系统软件内核的形式化验证工作：例如德国德累斯顿工业大学的VFiasco项目[6]、美国霍普金斯大学的EROS/Coyots项目[7]、德国的Verisoft项目[8-10]与后续的Verisoft-XT项目、澳大利亚国家通讯技术研究中心的L4.Verified项目[11]，以及微软研究院的众多项目，如Singularity[12]. VCC[13]、HAVOC [14]和Verve[15]等。

德国政府于2003年与2007年先后投入约两千七百万欧元，并联合多所大学启动了Verisoft（与后续的Verisoft-XT）项目来研究操作系统软件的验证，此项目还得到了宝马汽车、博世电气、奥迪汽车、德国电信等国际知名企业的支持与资助。Verisoft项目组提出了一个用来验证内核的验证框架CVM[8]，此框架不仅包含了处理器、内存、外部设备的形式化语义，还包含了一个经过验证的C0语言编译器。这样，由C0语言编写的内核代码与多个用户进程就可以通过Hoare逻辑在源语言级进行验证，通过编译器产生可靠的可执行代码。该项目组已经利用CVM验证框架验证了一个用于汽车电子控制单元嵌入式内核OLOS[9]与一个半虚拟化监视器PikeOS[10]。然而，这项工作未验证内核的启动与加载代码；CVM验证框架也仅能验证串行执行的简单内核，而不能验证并发执行的内核。

澳大利亚的L4.Verified项目组在2009年宣布成功验证了一个可以实际应用的操作系统内核seL4[11]，被验证的内核代码达8000行，其相关论文被当年计算机领域的顶级学术会议“The ACM Symposium on Operating Systems Principles（SOSP）”评为最佳论文，引起各方关注。与Verisoft不同，seL4只验证内核，而不考虑上层运行的用户程序；seLA也没有考虑C语言编译器的可靠性。最后值得一提的是，Verisoft和L4.Verified项目在验证过程中均提供机器可检查的正确性证明，从而使得人们不需要担心验证过程本身出错导致证明中出现错误。

微软研究院开发了一个研究型安全操作系统——Singularity[12]。其大部分代码通过C#语言的类型系统来保证安全性，但是有关任务调度、Sing#语言类型安全机制与垃圾收集的内核代码则需要另外进行验证[16]。微软研究院的另一个项目VCC开发了C语言程序验证工具[13]，用来自动验证虚拟机监视器——Microsoft Hyper-V。最近，微软完成了一个完整的小的实验性操作系统Verve的验证[15]。Verve的代码由两部分构成：类型安全的Sing#代码和汇编码。前者通过保持类型的编译器编译到类型安全的汇编码，后者的正确性则通过手工添加规约和使用自动定理证明器自动证明验证条件的方式验证。两者结合，保证了整个系统的类型安全和内存安全性，但该项目并未对Verve模块的功能正确性进行验证。

**2.2 相关工具和理论的开发**

近年来定理证明工具的进展为操作系统软件验证提供了重要支持。目前相关工具有两类：一类是人机交互的半自动化辅助证明工具，如法国INRIA开发的Coq[17]，英国剑桥大学和德国慕尼黑理工开发的Isabelle[18]等，另一类是自动定理证明工具，如微软开发的Z3[19]等。前者对证明自动化的支持较少，但优点是逻辑的表达力强，而且能够产生可供机器检查的机械证明，因而人们不用担心证明过程自身的错误。前述的德国的Verisoft项目[8]和澳大利亚的L4.verified项目[11]都使用Isabelle，并提供了机械证明。后者对逻辑的表达能力加以限定，可以完成自动化的定理证明，但并不能产生机械证明，因而我们需要相信定理证明器本身的正确性。微软Verve的验证[15]和垃圾收集器的验证[16]均依赖于Z3。近年来，已有工作试图将两者结合起来，将自动定理证明的技术引入到交互式证明工具中，同时产生可供机器检查的机械证明[20,21]。

随着各种验证工作的进步，人们逐渐认识到将各种验证技术和工具组合起来，构造完整的可信系统的重要性。耶鲁大学的邵中提出certified software的概念[22]，指出软件系统的验证不应简单相信底层的操作系统和运行环境，而是应诙将这些底层模块也一起验证，然后将各个经过验证的模块一起组成一个可信系统。普林斯顿大学的Andrew Appel开展了经过验证的软件工具链项目[23]，将静态分析工具、编译及优化工具、操作系统内核和库函数等分别验证并集成。INRIA的Xavier Leroy在2011年POPL的特邀报告上也提出了验证相关工具对软件验证的重要性[24]。

**2.3 我国在相关领域的工作**

我国在可信软件的研究上大部分偏向应用软件、网络软件以及软件执行环境可信性方面的研究。以基金委的重大研究计划“可信软件基础研究”为例，从所确立的四个核心科学问题及其论述[25]中可以看出这一点。这四个核心科学问题是：软件可信性度量、建模与预测，可信软件的构造与验证，可信软件的演化与控制，可信环境的构造与评估。从已经确定的重点项目来看，也是如此。这些重点项目有：基于环境建模的软件可信性需求获取和分析、大型分布式软件系统的行为监控与可信演算、面向软件可信性演进的软件测试技术研究、可信软件过程管理及风险控制模型和方法研究、基于虚拟机架构的可信计算环境与可信软件设计、基于网络环境的软件可信性建模与度量研究、面向C4KISR重大应用领域软件可信性需求分析有法与攻击性实验验证环境研究、基于认识与理解途径的软件可信性度量与评估体系及支撑技术研究、基于行为认证的电子银行可信软件验证实验环境研究。

从形式化验证领域来看，我国开展的研究工作多以算法、协议以及应用软件的验证为主。在操作系统软件验证方面也有初步工作，但仍处于起步阶段。

国内在形式化验证方向开展的工作主要包括模型检测和基于逻辑推理的验证两个方面，其中模型检测的领军人物之一是中国科学院软件所林惠民院士。对于软件验证来说，模型检测的优点是自动化程度高，但缺点是必须进行有限状态抽象、验证面临状态爆炸问题、以及缺乏软件本身和模型之间的一致性的验证。我们认为，模型检测适合对高层次算法、协议以及软件模型的检测和验证，该领域的理论和技术将会对系统软件验证、特别是对自动化工具的开发有很大指导意义，但对具体代码的验证的支持尚有较大不足，在基于逻辑推理的形式验证的研究方面，中国科学院软件所、清华大学、北京大学、中国科大、华东师范大学、上海交大、国防科大和南京大学等都开展了这方面的研究工作，包括验证理论、程序设计语言理论、出具证明编译、自动定理证明、自动化验证、以及各种具体算法、协议和软件的验证等大量工作。一方面，这些工作大多针对使用高级语言编写的应用层程序，或者仅仅对协议、算法等进行验证，并不直接针对底层系统软件代码；另一方面，这些工作为可信系统软件验证做了大量的理论积累和实践探索，其中的很多结果在系统软件验证中都可以得到借鉴或者直接应用。

我国针对操作系统软件本身的验证工作较少，可以说处于起步阶段，且大部分工作只是针对系统软件中的特定模块的特定性质的验证，对完整系统进行验证的工作较少。国防科大的杨学军、王戟等人对于操作系统内核程序的函数执行上下文进行验证[26]，并基于Assume-Cuarantee方法，对Linux操作系统中SSL协议的实现程序openssl-0.9. 6c进行了验证[27]；清华大学的董渊、王生原等人提出了一个基于多核的并行程序验证框架[28]，以及对字节码虚拟机的验证[29]。中科大-耶鲁高可信软件联合研究中心验证了系统软件中的一些关键模块，包括内存管理、垃圾收集、中断处理、线程切换等，并完成了一个小的操作系统内核雏形（约150行汇编代码）的验证[30]。华东师范大学何积丰院士承担的拔高基项目“汽车电子系统可靠性分析和验证方法研究”对汽车操作系统内核的安全性进行验证。中科院软件所的周巢尘院士正在开展对中国铁路控制系统CTCS-3的建模和验证工作。

**2.4 构造和验证高可信操作系统软件的挑战**

虽然国际上在操作系统软件及相关工具的验证中取得了较大突破，但完整的构造和验证系统软件仍然面临着很多挑战，其中的困难主要体现在以下几个方面：

（1）操作系统软件中大量的底层模块难以验证。操作系统中底层的很多模块直接和硬件交互，需要由汇编语言来实现。还有很多模块，如硬件中断处理和任务管理等涉及大量的并发和代码指针操作。现有的程序验证方法和理论还难以有效的对这些模块进行验证。例如，seL4内核中有多达1200行的汇编代码没有验证[11]，这些代码用于内核自启动、初始化与上下文切换等；为了避开验证复杂的中断处理功能，seL4内核通过轮询的方式来处理外部中断，与实际内核相差较大；为了避开代码指针推理，seL4内核中禁止出现函数指针与goto语句；seL4内核是串行执行而非并发；seL4的内存管理机制也没有验证。德国的Verisoft项目存在同样的问题：未验证内核的启动与加载代码；CVM验证框架[8]也仅能验证串行执行的简单内核，而不能验证并发执行的内核：内存管理机制虽然支持页异常的处理，但是虚拟内存管理功能过于简单，与实际内核差别较大。如何开发新的验证理论、方法和工具来对这些底层模块进行验证？

（2）系统复杂度高，不同的模块处于不同的抽象层次。从操作系统软件的层次式结构来看，低层模块为高层提供服务，同时将实现细节对高层屏蔽。不同抽象层次的代码处理不同的问题，具有十分不同的程序结构，甚至不同的程序设计语言来实现。比如处理硬件中断和任务调度的代码以及部分设备驱动代码可能用汇编语言和C完成，而上层模块则往往由C++甚至类型安全语言来实现。例如微软的Singularity系统[12]采用安全语言Sing#来实现高层模块，而底层内核中的大部分代码，例如任务调度、内存管理、进程间通讯管理与I/O管理等，则是由C或者汇编完成，安全性并无法保证。我们这里面临的问题是如何确保所有模块的可靠性，并同时保证这些模块构成一个完整系统后仍然具备可靠性？

（3）各模块可靠性需求的强度不完全一样。比如对于底层的任务调度模块，除了安全性（Safety）以外，我们可能还要考虑调度策略的公平性等性质。对于中断处理和设备驱动，我们则需要考虑与设备交互的实时性。而对于相对高层的垃圾收集器来说，其安全性则足我们考虑的主要因素，其他性质，如内存垃圾能否在规定的时间内收集，虽然也很重要，但验证需求并不如安全性急迫。一般来说，我们对可靠性的需求越往低层就越高，这是因为高层模块需要通过调用低层模块来实现其功能。低层模块出了问题通常会反映在高层模块上，影响高层模块的可靠性。从另外一个角度而言，低层模块相对代码量较小，哪怕保证其可靠性的代价相对较大，我们仍然能够承受。而高层功能性模块代码量较大，很多开销较大的严格保障可靠性的技术很难应用在这一层次。如何采用最合适的技术来处理不同层次的代码，得到不同模块所需要的可靠性保障，同时能够将这些技术集成，得到完整的可信系统？

（4）验证过程的效率和可靠性问题。在前述的Verisoft[8]、seL4[11]和CompCert[21]项目中，验证过程本身产生机器可以检查的正确性证明，使得我们不需要相信验证过程本身的可靠性。然而，这些项目的共同点是缺乏较好的工具支持，需要大量的手工验证，代价较大。而微软进行的垃圾收集的验证[16]和Verve系统的验证[15]采用了大量工具的支持，包括保持类型的编译器和自动定理证明器的支持，验证效率较高，但由于没有提供相应的可供检查的证据，我们需要相信这些复杂工具本身的实现是正确的，这影响了验证过程本身的可靠性。如何既获取工具的自动化支持，又不需要依赖于工具本身的正确性，是当前研究的一个热点问题。普林斯顿大学新近提出的软件工具链的验证以此为目标，但目前尚处于起步阶段[30]。

综上所述，由于操作系统软件在各类计算机应用中的核心地位，操作系统软件的验证已经得到国际学术界的广泛重视，并取得很大进展。但当前研究水平总体来说仍然处于探索阶段，尚面临很多挑战。我们希望能够抓住机遇，对这些问题尽快开展研究，以使我国在操作系统软件验证这一重要领域能迅速追赶并超过当前国际学术界的研究水平，建立国际领先地位；同时能将相关技术应用于实际，满足我国在高铁、航空、核电等安全攸关领域对可信系统软件的迫切需求。

**3 研究内容、目标和拟解决的关键科学问题**

**3.1 研究目标**

本项目对嵌入式操作系统软件的验证进行研究，在一个统一的框架下系统解决以下关键问题，为开发和验证可信操作系统软件提供方法、理论和工具的支持；并在该框架下完成一个完整的实用系统的验证。

（1）如何有效地控制操作系统软件的复杂度，使得我们可以有效掌握系统中各模块的接口以及模块之间的关系，并对操作系统进行模块化验证？

（2）如何开发好的验证方法和理论来验证现有技术不能验证的操作系统软件的复杂模块，如底层硬件相关的模块、含有大量并发或输入输出的模块等？

（3）需要哪些工具支持来提高验证的效率或者验证过程本身的可信性？

（4）如何重用现有的各种理论和工具，实现多种新老理论和工具的无缝结合？

（5）如何从验证中反过来对操作系统软件的开发提供启发和指导，以支持可信系统的开发？

**3.2 研究内容和关键科学问题**

作为一种解决上述问题的思路，我们这里建议从以下5个方面开展研究，最终提供一个操作系统软件的验证框架，并完成一个完整的实用操作系统的验证。

**（1）操作系统结构的形式化描述。**操作系统软件验证的主要困难在于其巨大的复杂度。AMD首席工程师Charles Moore曾建议，计算系统应应该向通信系统（如OSI模型的7层模型）学习，对操作系统的不同抽象层次进行划分和严格定义，这样将有助于控制系统复杂度。作为有效开发和验证操作系统软件的第一步，我们需要根据操作系统中各模块的功能以及彼此之间的交互，对操作系统的抽象层次进行总结和划分，并列各层次之间的接口进行形式化描述。

一方面，这种多层次的形式化描述可以为我们的验证提供模块化支持：我们对不同抽象层次的模块分别单独验证；根据各模块所在抽象层次的特性，我们可以选取最适当的理论和工具，而不需要考虑更底层的细节。另一方面，这种多层次模型还可以作为操作系统的开发、模拟（simulation/emulation）、测试和维护的指导性框架。例如，在划分抽象层次、定义层次间接口的时候，我们会发现操作系统结构的不合理性，并对其进行修改和调整。而新制定的层次式结构，则可以作为操作系统维护和下一次系统开发的基础。

这里要解决如下关键科学问题。首先，采用何种语言来对操作系统结构进行形式化描述？根据抽象层次的不同，很多时候我们对接口的规约也会有不同的要求。比如有些层次除了功能性的基本要求以外，还需要有对内存或者网络资源的消耗、实时响应能力等方面的规约。这就要求我们的规约语言要有足够的表达能力。其次，如何支持操作系统结构描述的可订制性和可扩展性？考虑到操作系统软件的多样性，以及集成第三方代码的要求，我们要求操作系统结构描述具有一定的开放性和可扩展性，并给出第三方代码的规约标准，为第三方代码开发、测试和验证提供依据。

**（2）新型程序设计语言和验证方法的开发。**如前所述，操作系统中处于不同抽象层次的模块需要处理不同的问题，而且具有十分不同的程序结构。我们认为，找到一种大一统的验证逻辑或方法以支持不同抽象层次上的所有模块的验证是很困难而且不现实的。相反，在对操作系统抽象层次的形式化描述的基础上，我们可以针对特定抽象层次，采用不同的编程语言和验证方法进行开发和验证，然后设法在一个基础框架上（见下述的研究内容4）将不同抽象层次上的经过验证的模块进行集成。这里我们将针对特定抽象层次的编程语言和验证方法称为领域专用语言和方法。

对于有些抽象层次，现有的语言和验证技术可以有效地重用（如可以采用类型安全的语言进行较高抽象层次模块的开发，以保证程序的Safety）。而有些层次，如前面所提到的底层硬件相关的层次，或者与外部交互事件较丰富的层次，现有的验证理论和技术还无法有效验证，同时这些层次中的很多程序错误是由于缺乏恰当的程序设计语言支持而造成的。我们需要设计新的领域专用编程语言和验证方法，为这些层次上的代码的开发和验证提供更加有效的支持。

这里要解决的关键问题除了克服特定抽象层次上所面临的理论和技术挑战外，还需要确保各种领域专用验证方法的模块化支持。简单来说，模块化验证要求我们支持各模块的独立验证。我们给出一个模块的规约并进行验证的时候，不应该需要知道系统中其它模块的细节，甚至不应该知道将来该模块会和其他哪些模块集成。这样，当我们对一个模块进行修改时，我们只需要修改对该模块的验证，而不会影响其他模块上的证明的有效性。

**（3）验证工具的开发和集成。**有效的开发和验证离不开相关工具的支持。我们一方面将研究现有各种工具的复用和集成，另一方面根据我们所涉及的领域专用逻辑和验证方法来开发专门的工具，包括不变式的自动推断、自动定理证明器、类型检查和静态分析工具、以及相关的开发和验证集成环境等。

这里所需要的另一个关键工具是能够在目标码产生证明的编译器，包括出具证明编译（Certifying Compiler）和经过验证的编译器（Certified Compiler）。前者根据源程序以及源级的验证，将程序和证明一起翻译到目标层；后者证明了编译过程本身的保语义性，从而确保了在源级经过验证的程序所产生的目标码的相关性质。有了目标码层次的证明，我们就可以将处干不同层次、使用不同方法验证的模块集成，得到完整的可信操作系统（见下述的研究内容4）。

这里要研究的关键问题包括：

* 如何使得验证方法从工具支持中获益，从而降低验证成本？有些验证工具和验证方法的结合比较直接，有些则不是那么直接。比如我们可否将目前广泛研究的各种静态分析技术与工具与Hoare逻辑结合，通过静态分析得到的信息来降低Hoare逻辑对自动定理证明能力的需求？
* 如何保证工具本身的正确性？要严格保证最终系统的安全性和正确性，我们必须选取正确（Sound）的工具，并对工具本身的正确性有所保证。通常保证工具的正确性有两种方式，一种是直接证明工具的正确性，另一种是对工具产生的结果进行检验（validation）。针对不同的工具，这两种方式的可行性和难易程度也将不同。

**（4）支持集成的基础语义和逻辑框架**。我们的最终目标是将经过单独验证的不同模块集成，得到完整的经过验证的可信操作系统。为此我们需要提供一个基础的语义和逻辑框架，该框架包含一个通用的语义模型，以及一个表达力强的元逻辑（meta-logic），使得我们在不同抽象层次所验证的不同性质都能在这个基础框架里描述。与领域专用逻辑不同，由于该框架并不直接用来书写程序规约（specification）并进行验证，因此与领域专用逻辑不同，易用性并不是这里关注的重点：通用性和表达力则是我们对基础逻辑框架的主要需求。

这里的关键问题是实现基础语义和逻辑框架的通用性。我们需要能够支持各种领域专用验证方法的集成。然而，不同的验证方法关注的性质不同，所采用的语义框架和规约语言（specification language）也会不同。比如有的验证方法可能会采用基于状态转换的操作语义，并使用分离逻辑断言作为规约语言，而有的方法会使用基于路径（trace）的语义，并使用时序逻辑书写规约。如何考虑到验证方法的多样性，给出一个通用的、表达能力强的基础语义模型和逻辑框架，使得各种力法均能集成到这个基础框架中来，是这里面临的主要问题。

**（5）完成实用的完整系统的验证**。上述的（1）、（2）、（3）、（4）四部分内容为我们提供了一个可信操作系统软件验证框架。基于该框架所提供的方法、理论和工具支持，我们需要完成一个完整的实用系统的验证，一方面来检验研究内容对实际系统的有效性，另一方面从具体验证中继续总结一般方法和理论，来改进我们的通用性框架。这里的关键问题是将上述方法、理论和工具应用于实际操作系统的各种具体问题，来考虑领域专用逻辑的设计、工具的开发和验证、不同模块的集成等等。

**4 结束语**

我国十分重视安全攸关软件和软件可信问题。例如，国家中长期科学和技术发展规划纲要（2006-2020）多次提到“高可信网络软件平台及大型应用支撑软件、中间件、嵌入式软件、网格计算平台与基础设施”和“高效可信的超级计算机系统、新一代服务器系统”等，国家自然科学基金委员会批准了重大研究计划“可信软件基础研究”，国家高技术发展(863)计划设立了专门的重大项目，研究高可信软件生产工具及集成环境，国家重点基础研究发展（973）计划将可信软件的研究确定为重点发展方向，研究基于网络的复杂软件可信度和服务质量。

如前所述，我国在可信软件的研究上大部分偏向应用软件、网络软件以及软件执行环境可信性方面的研究，对安全操作系统软件的构造和验证关注较少。除了前文中提到的国家自然科学基金支持的“可信软件基础研究”重大研究计划外，国家重点基础研究发展（973）计划中，有《基于网络的复杂软件可信度和服务质量及其开发方法和运行机理的基础研究》、《基于数学机械化方法的数字化设计制造与可信软件研宄》和《现代设计大型应用软件的可信性研究》等项目，这些项目基本上都不涉及底层操作系统软件的可信性研究。国家高技术发展（863）计划中设立的专门的重大项目《高可信软件生产工具及集成环境》，但也没有涉及操作系统软件的安全可信性问题。

嵌入式操作系统的验证是国际上的一大难题，本项目对嵌入式操作系统软件验证的方法、理论和工具开展研究，为应用软件提供一个高可信的基础计算平台，提高系统整体安全性和可靠性，适应对国家战略上对高安全操作系统软件的强烈需求。

**参考文献**

1. SamueL T. King, Peter M. Chen. Yi-Min Wang, et al., SubVirt: Implementing malware with virtual machines. In Proc. IEEE Symposium on Security and Privacy. pp 314-327,2006.
2. Bowen,J and Stavridou,V. Safety-critical systems. formal methods and standards.Software Engineering Journal. 8(4):189 - 209, 1993
3. Jean Souyri. Virginie Wiels. David Delmas. and Herve Delseny Formal Verification of Avionics Software Products. In FM 2009: Formal Methods. volume 5850 of LNCS,pages 532-546, 2009.
4. T.Hoare. The verifying compiler: a grand challenge for computing research. In Journal of the ACM (JACM).50(1). Pages 63-69. 2003. ACM Press.
5. T. Hoare, J. Misra. G..T. Leavens. and N.Shanka.The verified software initative: a manifesto.Avalible online at http://qpq.csl.sri.com/vsr/manifesto.pdf/view.
6. M. Hohmuth and H.Tews. The VFiasco approach for a verified operating system. In Proc. the 2nd ECOOP Workshop on Programming Languages and Operating Systems. Glasgow, UK. 2005.
7. J. S. Shapiro. M.S.Doerrie. E.NorLhup. S.Sridhar and M..Miller. Towards a verified. general-purpose operating system kernel. In Proc. the NICTA Formal Methods Workshop on Operating Systems Verification 2004. Technical Report 0401005T-I, NICTA. Sydney. Australia. 2004.
8. T.Rieden and A. Tsvban. CVM ~ A verified framework for microkernel programmers In Proc. 3rd International Workshop on Systems Software Verification {SSWOS). volume 217C of ENTCS, pages 151-168.Elsevier Science B.V. 2008.
9. M. Daum. N. W. Schirmer and M. Schmidt In Proc. 7th International Conference on Software Engineering and Formal Methods (SEFM'09). pages 23-32.Hanoi. Vietnam. IEEE Computer Society. 2009.
10. C. Baumann and T. Bormer. Verifying the PikeOS microkernel: First results in the Verisoft XT Avionics project In Proc. Doctoral Symposium on Systems Software Verification (SSV 2009) . pages 20-22. June 2009.
11. G. Klein. K. Elphinstone et. al. seL4: Formal verification of an OS kernel. In Proc. the 22nd ACM Symposium on Operating Systems Principle (SOSP '09) . Big Sky, MT. USA Oct. 2009.
12. G. C. Hunt and J.R..Larus Singularity: Rethinking the software stack. In ACM SIGOPS Operating Systems Review. vol. 41. no. 2. pp. 37-49, ACM. April 2007.
13. M. Dahlweid M.Moskal, T.SanLen. S.Tobies. and Wolfram Schulte. VCC: Contract-based modular verification of concurrent C In Proc. 3ist International Conference on Software Engineering (ICSE '09). IEEE Computer Society, 2008.
14. S. K Lahiri. S. Qadeer. and Z. Rakamaric. Static and precise of concurrency errors in systems using SMT solvers. In Computer Aided Verification (CAV09) . Springer Verlag, February 2009.
15. Jean Yang. Chris Hawblitzel. Safe LO the last in system. In Proc. 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI2010), pp 99-110. 2010.
16. C.Hawblitzel and E. Petrank. Automated verification of pratical garbage collectors.　In Proc the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '09) . ACM. New York,NY, USA. 441-453. 2009.
17. B. Barras et al. The Coq Proof Assistant reference manual Technical report. INRIA, 1998.
18. Tobias Nipkow. Lawrence C. Paulson. Markus Wenzel: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. Springer. 2002.
19. J Leonardo de Moura and Nikolaj Bjorner. Z3: An Efficient SMT Solver. In Proc Conference on Tools and Algorithms for the Construction and Analysis of Systems ( TA CAS 2008) . Budapest. Hungary, 2008.
20. Sascha Bohme. Tjark Weber: Fast LCF-Style Proof Reconstruction for 23. ITP 2010; pp 179-194.
21. Antonis Stampoulis, Zhong Shao. VeriML: typed computation of logical terms inside a language with effects In ICFP 2010: 333-344.
22. Zhong Shao. Certified software. Commun. ACM 53(12). pp 56-66, 2010.
23. Andrew W.Appel. Verified Software Toolchain. In Proc. 20th European Symposium on Programming (ESOP 2011), to appear. March 2011.
24. Xavier Leroy. Verified squared: does critical software deserve verified tools? In Proc. the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 2011). pp 1-2. 2011.
25. 刘克、单志广、王戟、何积丰、张兆田、秦玉文，“可信软件基础研究”重大研究计划综述，学科进展与展望，2008年第3期.
26. 江黎、杨学军、王戟、罗宇操作系统内核程序函数执行上下文的自动检验软件学报，18 (4), 2007.
27. 项森、陈意云、林春晓、李隆动态存储管理安全验证的Coq实现计算机研究与发展44 C2): 361-367，2007.
28. 朱允敏、张丽伟、王生原、董渊、张素琴面向多核处理器的低级并行程序验证电子学报，37 (z1), 2009.
29. Yuan Dong, Kai Ren. Shengyuan Wang and Suqin Zhang. Certify Once, Trust Anywhere: Modular Certification of Bytecode Programs for Certified Virtual Machine. In APLAS 2009: 275-293.
30. X.Feng,Z Shao.Y Guo and Y Dong. Combining Domain-Specific and Foundational Logics to Verify Complete Software Systems. In Proc. VSITE2008..LNCS Vol..5295. pages 54-69. Springer-Verlag. 2008.