微内核完整性保障研究与应用

于淑英 黄 皓 刘国斌

(南京大学计算机科学与技术系 软件新技术国家重点实验室 南京 210093)

摘 要 为了避免安全操作系统中访问控制机制被篡改、绕过,提出利用微内核多服务器结构为安全核完整性提供保障。微内核提供的进程隔离和消息传递机制使得各个服务进程独立运行,通过受控的消息机制进行交互,有效保证了各个模块的完整性。微内核结构的简单性和模块化为形式化验证奠定了基础。原型系统 Nutos 利用 Flask 安全体系结构为用户提供灵活多策略的强制访问控制,由微内核多服务器结构为 Flask 中的安全服务器和引用监控器提供完整性保障,确保安全策略的正确实施。

关键词 完整性,安全保障、微内核多服务器结构,进程隔离,消息通信

Integrity Assurance of Micro-kernel Operating System and its Application

YU Shu-ying HUANG Hao LIU Guo-bin

(State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract To avoid the security mechanism applied in operating systems being bypassed or tampered, this paper proposed the use of micro-kernel, multiserver architecture to assure the integrity of security kernel. Process isolation and message passing provided by the micro-kernel make the processes above isolated and protect the integrity of them effectively. Simplicity and modularity, the most obvious advantages of micro-kernel, laid an excellent base for the future formal verification. The prototype operating system, Nutos, was presented as an example on how to use these mechanisms to enforce security. It combined the multiserver architecture and the Flask security infrastructure to provide for flexibility in security policies and integrity assurance for security sever and reference monitor.

Keywords Integrity, Security assurance, Micro-kernel multi-server architecture, Process isolation, Message passing

1 概述

计算机应用领域的拓展,使得信息安全已经成为计算机领域的一个重要研究课题。操作系统安全是计算机安全的重要基础,要解决广泛的计算机安全问题,必须开发出能够满足实际应用需求的安全操作系统。安全操作系统的研究从20世纪60年代就已经开始[14],早期安全操作系统的设计目标主要是将特定的安全策略应用在系统的访问控制中或者按照诸如 TCSEC 等标准去实现固定的安全需求,如安全增强的 Multics 把 BLP 模型应用到 Multics 系统中[1];System V/MLS 多级安全操作系统[2]以 TCSEC 标准的安全等级 B 为设计目标。近些年,为了满足各种用户不同的安全需求,对安全策略的灵活支持已成为安全操作系统的主要设计目标,如 Fluke 项目[4]和 SE-linux[5]分别在微内核和单内核系统上实现了支持策略灵活性的 Flask 安全体系结构[10],提供灵活多策略的访问控制。

在研究系统中使用什么样的安全策略和如何在系统中实施这样的安全策略的过程中,引用监控机的思想被广泛应用在安全操作系统的设计中。引用监控机负责控制系统中程序对资源访问,它保证程序对资源的所有引用都得到授权机制的仲裁,它小而简单,易于验证其正确性和功能性。这一思想的提出者 J. P Anderson 在引用监控机的基础上又提出了安全核的概念[6]。安全核是系统中与安全的实现有关的部分,包括引用监控机制、授权机制和授权管理机制,它必须具有自我保护能力,即使受到攻击也能保持自身的完整性。由此可见,安全操作系统提供的安全服务、达到的安全目标就是由安全核正确性和完整性决定的。以前的安全操作系统通过合理的系统设计可以使引用监控机制非常简单,再利用形式化验证的方式证明其功能性和正确性。但是对于安全核完整性保障的研究比较少,也还没有比较好的解决方案。Fluke 系统利用递归虚拟机[15]和权能系统的机制为安全核提供完整性保障,但是导致系统性能严重下降。而其它的安全操作系统,如 SE-linux,甚至没有将安全保障作为研究内容,引用监控器的完整性依赖于自身实施的安全策略的保护。鉴于这种原因,本文的研究着眼于从系统结构角度保证安全核的完整性,为安全策略的正确实施提供安全保障。

本文第2节根据安全核的概念提出安全操作系统完整性保障的需求,第3节阐述微内核结构在保障安全核完整性方

到稿日期:2008-01-29 本课题得到国家自然科学基金(60473093)资助。

于淑英(1983-),女,硕士研究生,研究方向为操作系统安全,E-mail:shuyingyu@gmail.com;黄 皓(1957-),男,教授,博士生导师,研究领域为计算机网络安全;刘国斌(1982-),男,硕士研究生,研究方向为操作系统安全。

面的作用，第 4 节说明原型系统 Nutos 的设计实现及与相关工作的比较，最后给出了结论。

2 完整性保障

1983 年，美国国防部颁布了可信计算机系统评价标准[16]，简称 TCSEC。标准中使用的可信计算基(TCB: Trusted Computing Base)是在基于安全核技术的安全操作系统研究的基础上提出来的。TCB 即一个计算机系统中的保护机制的全体，它们共同负责实施一个安全策略，包括硬件、固件和软件；一个 TCB 由在一个产品或系统上共同实施一个统一的安全策略的一个或多个组件构成。TCSEC 中重新给出了安全核的定义：一个 TCB 中实现引用监控机思想的硬件、固件和软件成分；它必须仲裁所有访问，必须保护自身免受修改，必须能被验证是正确的。

操作系统功能是由很多逻辑模块共同完成的，文件管理、进程管理、内存管理等功能模块为用户提供资源管理服务；访问控制机制为用户提供安全服务，在不同的安全操作系统中有不同的实现。另外操作系统内核中还包括了设备开发商等提供的第三方软件模块，主要是驱动模块，控制相应硬件设备资源。根据安全核的定义可知，安全核应该包括访问控制机制以及文件管理、进程管理和内存管理等模块中与安全相关的部分。保证这些模块的完整性，就是保障安全核的完整性。

保证一个系统不会被恶意的破坏导致崩溃或者影响到正常服务，是对一个软件系统的完整性要求[3]。用户通过程序对资源数据进行操作而获得服务，用户得到的服务结果取决于程序对资源进行何种操作，所以服务正确与否取决于对资源实施操作的程序的正确性，以及程序与资源之间操作关系的正确性。这些是由程序执行的代码以及程序依赖数据决定的。软件的完整性是否被破坏，软件所依赖的数据是否遭受攻击等问题，在一般情况下是一个不可判定问题[8]。但是我们要研究的安全核是操作系统模块，这些模块代码质量能得到很好的保证，我们的研究基于这样的假设：安全核模块代码都是正确的，安全核根据正确的代码对依赖数据实施的操作也是正确的。那么安全核提供的安全服务的正确性决定于这些模块运行过程中代码和数据的正确性。如何使得安全核模块在运行时保持代码完整性，避免依赖数据免受攻击，就是本文关注的问题。

综上所述，安全操作系统完整性保障需求就是保护安全核的完整性。进一步讲，就是在安全核模块代码正确的基础上，保证安全核的代码和依赖数据免受攻击。

将安全相关的功能隔离起来是一种有效的安全程序设计原则。本文将这种原则应用在操作系统安全核的完整性保护上，设计出一种能够有效保证安全核模块代码和数据完整性的系统结构。

3 系统结构设计

操作系统中各功能模块虽然关系紧密但功能相对独立，它们有自己的数据结构且相互间有比较明确的调用接口。但在 unix，linux 等传统操作系统中，模块之间紧密耦合，运行在同一地址空间内，相互之间可以直接调用函数并且访问数据，都运行于内核态，拥有对系统的所有特权。安全核没有得到隔离，如系统中的自主访问控制、SE-linux 提供的安全服务器模块同所有其它模块运行在一起，可以被其他模块任意修改。要保证单内核系统中安全核的完整性，就必须要保证整个单内核的正确性和完整性。但是在内核中运行的驱动等第三方软件正确性不能保证的情况下，获得整个内核完整性是不可行的。单内核模块完整性难以保证是由单内核本身的结构导致的，所以我们放弃这种传统的操作系统结构，采用微内核多服务器系统结构。

3.1 微内核多服务器结构

微内核多服务器结构是为了增强系统的可扩展性提出的，也有效提高了系统的可靠性[9,12]。系统结构如图 1 所示。微内核使内核最小化，只包含必要的机制而不包含策略。内核只实现中断处理、进程地址空间隔离、进程间通信和进程调度。功能模块如内存管理、进程管理、资源管理和驱动程序等都运行在内核之外，作为单独的服务进程在用户模式下运行。服务进程不具有特权，只拥有完成任务所必需的最小权限。所有的特权都限制在微内核中。微内核提供了一系列内核调用给服务进程使用，帮助服务进程完成特权级操作，例如，驱动程序不能直接操作 I/O，需请求内核调用 devio 完成。服务进程和应用进程之间通信都是通过内核控制的消息机制来完成的。

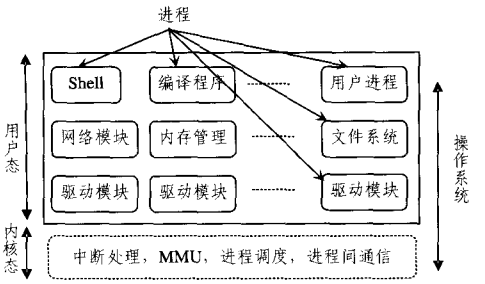

图 1 微内核多服务器系统结构图

3.2 完整性保障分析

微内核多服务器结构在设计上满足了以下几个特点：简单、隔离、职责分离和最小特权、容错。这几点不仅有效保证操作系统的可扩展性和可靠性[7]，而且为系统中各个模块的完整性提供了保障。

首先，微内核短小，功能简单，易于保证其正确性和功能性。代码越少意味着错误就越少。微内核代码量很少，有效减少了内核中可能存在的错误或者漏洞，而且小内核比较容易维护，一个人就可以掌握整个内核，这样比较容易发现错误。另外，微内核功能简单，适宜形式化验证其功能正确性和完备性。用形式化方法验证微内核功能正确性是项目中的另外一个研究课题，已取得一定的进展。本文基于这种假设：微内核的正确性和完整性都能得到保证。

其次，进程隔离机制保证了每个进程的独立完整性。微内核的进程隔离机制使系统各功能模块作为独立的服务进程实现，由 MMU 硬件和保存在微内核中的段页表保证了进程物理地址空间上的隔离，使得直接跨进程的读写操作在硬件级上被禁止。除了进程本身和微内核，其他进程没有能力对数据段和代码段进行操作，有效保证了代码段、数据段和堆栈段的完整性。

另外，进程之间只能通过受控的消息机制交互，避免了直接函数调用导致的缓冲区溢出等问题。消息机制中，消息的

大小是固定的。当进程间需要大量的数据交互时，由消息来指明交互的数据缓冲区，通过内核提供的 copy 和 vcopy 服务实现进程间缓冲区数据的拷贝。系统中只提供同步的消息机制，使得接收进程能够主动控制对自己消息缓冲区的写操作。另外，大数据块的传递通过内核服务来实现，而内核中规定了只有服务进程具有 copy 和 vcopy 的权力，保证了服务进程对于大数据缓冲区读写的主动权。由此可见，无论接收消息还是接收缓冲区数据，对于服务进程都是主动行为。这样服务进程在接收到外来数据过程中，主动控制了对自己缓冲区的写入，防止了缓冲区溢出对堆栈段数据的破坏。

最后，职责分离和最小特权两个重要的安全原则在系统中得到应用。系统中所有的特权操作都在微内核中执行，服务进程等通过内核调用请求服务。这样内核不但可以控制特权操作的执行，还可以控制每个进程允许向内核请求哪些内核调用。在微内核中，由内核调用位图限制进程允许的内核调用集合和允许的 I/O 操作端口。微内核以进程为单位，分配给每个进程足够完成工作的最小权限集合。这样避免了某个进程具有过度集中的权限，当进程被攻击时造成严重破坏。例如，微内核可以禁止打印驱动写用户地址空间、读写硬盘 I/O 端口。同样，微内核只允许内存管理程序请求对进程段页表的修改，防止其他进程恶意破坏进程隔离机制。

综上所述，微内核多服务器结构将系统中各模块作为独立的进程隔离开来，限制进程间只能利用被内核控制的消息通信，使得进程的代码和数据保护在进程的私有空间中，不会受到其他进程的非法破坏，从而保证了各模块的代码和数据完整性。另外，系统中所提供的职责分离和最小特权原则，限制了每个进程的最大权力，有效限制了攻击的破坏力度。

4 原型设计和实现

由于以往安全操作系统大多没有为安全核提供完整性保障，我们项目的目标是建立一个这样的安全操作系统：它能够为用户提供细粒度的、灵活多策略的可靠强制访问控制，并且为安全目标的实现提供安全保障。这个系统结构要尽可能的简单，以便能对其完整性保障功能进行形式化验证，使得系统中的安全服务得到保证。同时我们的系统面向安全专用环境，如保密单位的单向网关系统。因此我们采用了微内核多服务器结构，并在系统中实施了强制访问控制机制，在 Minix3.0 的基础上开发了我们的安全操作系统 Nutos。系统结构如图 2 所示。

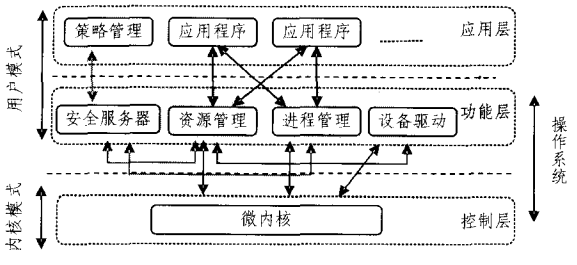

图 2 Nutos 系统结构图

整个系统划分为 3 层：控制层、功能层和应用层。控制层只包含微内核，运行于内核模式。功能层包括资源管理、进程管理、网络管理、安全服务器和驱动程序等服务进程。它们向用户提供 posix 标准的系统调用服务。应用层包括运行在操作系统之上的各种用户程序，与单内核中的用户态进程相同。功能层和应用层都运行在用户模式下。其中根据 flask[10] 体系结构设计了为用户提供安全服务的安全服务器和引用监控器：安全服务器负责维护系统中所有与强制访问控制相关的安全策略数据和主体安全属性信息，并负责对请求的操作做出安全仲裁。引用监控器添在资源管理进程、进程管理进程中负责向安全服务器请求服务仲裁并根据结果执行或拒绝请求服务。系统中的网络、硬盘、打印机等硬件设备都抽象为一种资源，由资源管理器统一实施对它们的访问控制。因此安全服务器、资源管理和进程管理构成了此系统中的安全核，依赖它们为用户提供正确可靠的安全服务，保护用户的资源和系统的其他模块免遭攻击。

4.1 隔离保护的安全核

Nutos 系统中微内核多服务器系统结构通过微内核提供进程隔离和消息机制将各个功能模块独立实现为进程，进程之间只能通过消息机制互相通信。微内核控制使普通用户进程只能向资源管理进程和进程管理进程请求系统调用，然后由资源管理或者进程管理中的引用监控机制向安全服务器请求仲裁。与功能层的驱动程序或者网络管理的交互，由资源管理器来负责。因为 C 语言缺少参数类型检查会带来安全方面的问题，如缓冲区溢出，Nutos 系统中在资源管理和进程管理模块实现了对参数进行严格检查的机制，称为安全接口。它是一段经过严格审查的代码，由接受消息的函数触发，对接受的数据进行合法化检查，防止恶意的参数对进程的私有数据造成破坏。这样由安全服务器、资源管理器和进程管理器组成的安全核形成一个相对隔离的区域，由资源管理和进程管理通过安全的消息机制与外界交互，如图 3 所示。这样使得在系统运行过程中，安全核进程的代码和数据都不会遭到攻击或者意外破坏，有效保证了安全核的完整性，为安全策略的正确实施提供了保障。

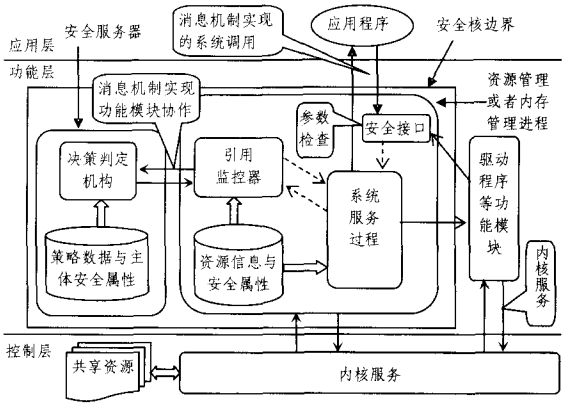

图 3 Nutos 系统安全核组成及交互

4.2 策略管理

Nutos 系统的目的要为用户提供灵活多策略的安全服务，所以必须为用户提供策略配置的接口，使用户能够配置满足自己的安全需求的策略。图 2 中位于应用层的策略管理进程就是为用户提供的策略配置工具，它通过安全服务器提供的策略更新接口来更新安全策略。但是策略数据是安全核不可缺少的一部分，是安全服务器做出仲裁的依据，其完整性是必须的。策略数据配置好之后保存在安全服务器中，内核提供的最小特权原则保证其他进程不能请求安全服务器的策略

更新服务，所以保护的重点在策略管理进程不被非法执行。为了达到这个目的，我们为系统增加了策略管理员这一特别角色，将策略数据和策略管理程序放在系统中的/secure文件夹下，只有策略管理员可以读写这个文件夹，对其他用户都不可见，这一原则在引用监控器的代码实现中得到保证，避免了对安全策略的依赖。这样有效防止了攻击者获得普通管理员权限后破坏策略数据。当然在某些安全性要求比较高的情况下，可以实施USBKey激活策略管理等更加严格的保护措施来保证策略管理员账户的安全。将策略管理职责独立出来为实施其他的保护措施提供了方便。

4.3 性能测试与分析

Nutos系统采用了微内核结构，早在20多年前就被提出，但是由于性能比较差，一直没有得到推广。但是第二代微内核技术在IPC上有了很大的改进，使得微内核性能下降很少，L4linux比linux的性能下降2%～3%[11]。虽然在Nutos系统中还没有达到这样的性能，但是这却说明微内核技术的性能已经不是瓶颈。我们进一步利用引用监控器和安全服务器间多消息捆绑发送的方法保证Nutos系统性能不会下降很多。表1为我们在P4 CPU 1.70GHz，256M内存，40GB IDE硬盘的硬件环境下的Minix3.0和Nutos相关系统调用的速度的测试结果。由表1可以看出，Nutos系统与Minix3.0相比较，性能损失在10%以内，这是由于系统运行时增加了对安全服务器的决策请求的消息交互造成的。另外，从表中还可以看出，对于大数据量的读写，性能几乎没有下降。可见，微内核和flask结构导致的性能损失可控制在15%以内。这种性能下降所换取的系统的安全性对于安全性要求比较高的非普通用户是完全可以接受的。

表1 系统调用性能比较(单位：微秒)

系统调用	Minix 3.0	Nutos	比率
open+close	3.914	4.215	1.077
write(1K)	4.795	5.064	1.056
read(1K)	3.418	3.626	1.061
read(1M)	3061.9	3068.0	1.002
lseek	1.751	1.823	1.041

4.4 相关工作比较

第1节中提到在安全操作系统发展过程中，像Nutos系统一样将安全保障作为设计目标的还有Fluke系统。Fluke[4]项目是美国安全计算公司(SCC)和国家安全局(NSA)开发的DTOS系统的后继。Flask体系结构就是Fluke操作系统的抽象模型。该项目目标分为两个方面：一方面是安全性目标即建立一个策略灵活的访问控制模型，支持动态安全策略；另一方面是保障能力目标即通过运用形式化描述和推理手段实现对关键安全功能的验证。项目的研究工作主要集中在安全策略的实施机制和这些机制与安全策略间的协调作用。Fluke系统结构基于递归虚拟机[15]思想和权能系统的基本机制。硬件之上的微内核提供像传统的虚拟机一样的硬件虚拟功能，支持上层递归虚拟机的运行。这里的递归虚拟机模拟程序运行需要的基本指令集、底层系统调用API和进程间通信API，支持它上层递归虚拟机的运行。每个进程运行在一个单独的递归虚拟机内，形成嵌套进程，进程之间只能通过虚拟机间接通信，并由虚拟机控制进程的权能。从而由递归虚拟机实现进程之间的全隔离。这样每运行一个进程会额外运行一个虚拟机，虽然保证了安全相关进程的隔离性，但是导致资源的大量浪费，使得系统性能下降0%～30%不等。Nutos系统利用多服务器结构实现进程之间的隔离避免了性能的大幅度损失。另外，递归虚拟机不仅实现IPC，管理它们的权能，还要模拟进程运行需要的指令集合，增加了递归虚拟机的复杂性，从而对形式化验证造成影响。Nutos系统利用微内核保证了系统的简单性，为形式化验证各个模块的隔离性提供了条件。

结束语 本系统的设计利用了微内核多服务器系统结构来保证安全操作系统中安全核的完整性。通过改变操作系统的系统结构，避免单内核系统中安全核与不可信模块紧密耦合、运行在同一地址空间中的缺点，安全核模块作为单独进程运行，有效地保证了安全相关机制的完整性，保证了系统安全服务的正确可靠性。另外，本系统遵从了简单模块化的设计原则，为进一步形式化验证安全核的完整性奠定了基础。利用形式化方式验证系统中的微内核的完备性、安全核的完整性是本项目的另一部分的研究工作，其中对微内核的验证已取得很大进展。

Nutos的目的是为用户提供细粒度的、灵活多策略的可靠强制访问控制。系统为实施访问控制的安全服务器和引用监控器提供了安全保障，这就为基于此的用户安全策略的正确实施奠定了基础。下一步的工作是研究什么样的安全策略能够满足用户的安全需求，以及安全服务器中多策略的复合问题。微内核多服务器结构为安全核提供完整性保障，安全核为上层应用程序提供安全服务，这样由底向上逐层保护最终形成有纵深的防御体系，在不同强度的攻击下尽可能多地保证系统各种安全底限不被破坏。

参 考 文 献

[1] Bell D E, LaPadula L J. Secure Computer System: Unified Exposition and MULTICS Interpretation. MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA, USA, Mar. 1976

[2] Flink II C W, Weiss J D. System V/MLS Labeling and Mandatory Policy Alternatives. AT&T Technical Journal, May/June 1988: 53-64

[3] Limoges C G, Nelson R R, Heimann J H, et al. Versatile integrity and security environment (VISE) for computer systems // Proceedings of the 1994 Workshop on New Security Paradigms. Little Compton, Rhode Island, 1994: 109-118

[4] Secure Computing Corporation. Assurance in the Fluke Microkernel: Formal Security Policy Model. CDRL Sequence No. A003, Secure Computing Corporation, 2675 Long Lake Road, Roseville, Minnesota 55113, Feb. 1999

[5] Loscocco P, Smalley S. Integrating Flexible Support for Security Policies into the Linux Operating System. Technical report. NSA and NAI labs, Jan. 2001

[6] Anderson J P. Computer Security Technology Planning Study. Volume II. ESD-TR-73-51, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA, USA, Oct. 1972

[7] Herder N, Bos H, Tanenbaum A S. A lightweight method for building reliable operating systems despite unreliable device drivers. Technical Report IR-CS-018. Jan. 2006

[8] Bisop M. Computer Security: Art and Science. ISBN: 0201440997. Copyright @ 2003

[9] Liedtke J. Toward real microkernel. comm. of ACM, 1996
[10] Spencer R, Smalley S, Hibler M, et al. The Flask Security Architecture: System Support for Diverse Security Policies // Proceedings of the 8th USENIX Security Symposium. 1999: 123-139
[11] Chou A, Yang J, Chelf B, et al. An Empirical Study of Operating System Errors // Proc. 18th ACM Symp. on Oper. Syst. Prin. 2001: 73-88
[12] Swift M, Annamalai M, Bershad B, et al. Recovering Device Drivers // Proc. Sixth Symp. On Oper. Syst. Design and Impl. 2004: 1-15
[13] CSC-STD－001-83. Department of Defense Standard, Department of Defense Trusted Computer System Evaluation Criteria, DoD Computer Security Center, Aug. 1983
[14] 石文昌. 安全操作系统研究的发展. 计算机科学, 2002, 29(6): 29(7)
[15] Ford B, Hibler M, Lepreau J, et al. Microkernels meet recursive virtual machines // Proceedings of the Symposium on Operating Systems Design and Implementations. New York: ACM Press, 1996: 137-151
[16] Anderson J P. Computer Security Technology Planning Study Volume II. ESD-TR-73-51, Vol. II, Electronic Systems Division. Air Force Systems Command, Hanscom Field, Bedford, MA, USA, Oct. 1972

(上接第 243 页)

陷数据的统计分析功能。须实现的最基本的统计功能是计算不同缺陷属性值所对应的缺陷数量及其百分比，统计结果要能够以表格和直方图(或饼状图)显示，从而可以明显地展示出缺陷相对于各属性值的分布，这种分布反映了导致软件缺陷的过程问题。这样，在缺陷管理工具的支持下，缺陷数据在软件开发过程中被实时地录入并实时地统计分析，将产品质量情况和软件过程问题快速地反馈给开发人员，使其采取相应的调整措施，减少缺陷的引入。

软件组织在不同的成熟度阶段，以及在开发不同领域的软件产品时，产生的缺陷信息会有所不同，缺陷的原因也会有所差别，这就要求缺陷管理工具具有缺陷属性及其值的自定义功能[12]。

基于缺陷分类的定量分析方法以客观的数据统计为基础，易于实现自动化和定量的控制，实时性好，分析覆盖率高，资源消耗小，适用于大型项目。但该方法的局限性在于缺陷数据的统计结果难以揭示有关人员和管理因素的深层次问题，往往只是这些问题的间接"反映"，因此通常还需结合定性方法作为补充。例如，如果通过统计发现某类型的缺陷相对于各开发阶段有明显异常的数量分布趋势，要对这种情况做进一步的定性分析，发现开发阶段中存在的有关人员、管理或技术上的深层次原因。此外，该方法在提高了分析抽象程度的同时，也容易忽略对一些特殊情况和个别严重缺陷的详细分析，有可能遗漏一些特定的缺陷原因。

结束语 软件工程不同于传统的工程学科，没有任何一种模型可以用来描述所有软件开发过程，也不可能完全实现自动化和定量的工程控制，在很多情况下开发人员必须根据当前项目的实际情况并结合经验数据做出主观判断。因此，软件组织在分析缺陷原因时，应融合定性方法和定量方法，把客观的数据统计与主观判断结合起来，具体方法是：

第一，在做定量分析时，结合定性方法得到有关人员、管理等方面的深层次原因和有效的解决方案；在做定性分析时根据需要收集数据，以量化的统计结果作为分析的依据和结果的验证。

第二，用定量的方法分析缺陷的宏观趋势，由此发现的共性问题再采用定性的方法进行具体分析。此外，对于会造成严重后果的缺陷，应采用定性的方法(如失效模式和影响分析、故障树分析等)进行逐个详细分析，以预防该类缺陷的产生。

在"能力成熟度模型集成(CMMI)"中，缺陷原因分析属于 level 5 的"原因分析与解决(CAR)"关键过程域，这在一定程度上给人们造成了一种印象，即只有成熟度级别很高、已完全实现了量化管理的软件组织才能进行缺陷原因分析和预防，实际上这是一种错误认识。在缺陷管理工具的支持下，将定性与定量方法相结合，处在较低成熟度级别(level 2 以上)的软件组织也可以分析缺陷的产生原因并采取相应的改进和预防措施[8]。在今后的研究工作中，我们将继续对缺陷原因分析的方法及其支持工具进行深入研究，使之更为实用和易用。

参 考 文 献

[1] 卡耐基梅隆大学软件工程研究所. 能力成熟度模型(CMM)：软件过程改进指南. 刘孟仁，等译. 北京：电子工业出版社，2001
[2] Rooney J J, Heuvel L N V. Root Cause Analysis for Beginners. Quality Progress, July 2004
[3] Doggett A M. A Statistical Comparison of Three Root Cause Analysis Tools. Journal of Industrial Technology, 2004, 20(2)
[4] U. S. Department of Energy. Root Cause Analysis Guidance Document. http://www.hss.doe.gov/NuclearSafety/techstds/standard/nst1004/nst1004.pdf, February 1992
[5] Fenton N, Pfleeger S L. 软件度量-严格而实用的方法. 第 2 版. 北京：机械工业出版社，2004
[6] Lanubile F, Shull F, Basili V. Experimenting with Error Abstraction in Requirements Documents // Proceedings of 5th International Symposium on Software Metrics. Bethesda, Maryland, 1998
[7] Leszak M, Perry D E, Stoll D. A Case Study on Root Cause Analysis // Proceedings of the 22nd International Conference on Software Engineering. Limerick, Ireland, 2000
[8] Mays R G, Jones C L, Holloway G J, et al. Experiences with Defect Prevention. IBM Systems Journal, 1990, 29 (1)
[9] Buglione L, Abran A. Introducing Root-cause Analysis and Orthogonal Defect Classification at Lower CMMI Maturity Levels // Proceedings of 1st International Conference on Software Process and Product Measurement. Cadiz, Spain, 2006
[10] Dunn A. Getting Root Cause Analysis to Work for You // Proceedings of International Conference of Maintenance Societies (ICOMS). Sydney, Australia, 2004
[11] Chillarege R. ODC-a 10x for Root Cause Analysis // Proceedings of RAM 2006 Workshop. Berkeley California, 2006
[12] 刘海，郝克刚. 软件缺陷数据的定义. 计算机应用，2008(1)
[13] IBM Research Center for Software Engineering. Orthogonal Defect Classification. http://www.research.ibm.com/softeng/ODC/ODC.HTM, 2002