基于代码路径的安全操作系统性能优化方法①

王 硕[1,2,3], 杨秋松[1,2], 吴 涛[1,2,3]

[1](中国科学院软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院软件研究所 计算机科学国家重点实验室, 北京 100190)
[3](中国科学院大学, 北京 100049)

摘 要: 为了满足面向访问验证保护级的要求, 研发新一代高等级安全操作系统, 我们采用微内核的架构设计和实现了面向访问验证保护级的安全操作系统原型系统(VSOS), 并通过设计和实现新的访问监控器来满足安全内核设计原则中的不可旁过和总是被调用两项要求, 但访问监控器的引入导致 VSOS 的性能产生较大的损耗. 提出了一种基于代码路径优化的方法, 用于改进访问监控器的实现和调用方式, 以及可信路径机制的实现方式. 实验表明, 通过此方法 VSOS 的性能和可信路径过程的用户体验都得到了提升.

关键词: 安全操作系统; 代码路径; 访问监控器; 性能优化; IPC 重定向

Code Path-Based Optimization Method of Secure Operaing System

WANG Shuo[1,2,3], YANG Qiu-Song[1,2], WU Tao[1,2,3]

[1](Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[3](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: To satisfy the requirement of access verification protection level for secure operating system and development the next generation secure operating system with high security levels, we design and implement the Verification-oriented Secure Operating System prototype (VSOS) in microkernel architecture. VSOS meets the two of the principles of designing security kernel, which suggest that reference monitor must be tamper proof and always be involved, by designing and implementing the new reference monitor. However, the introducing of reference monitor causes great performance penalty on VSOS. In order to improve the performance of the VSOS, a code path-based optimization method is used to improve the way of implementing and calling reference monitor as well as implementation of the security mechanism such as trusted path. Experiment results demonstrate that both the performance and the user experience of VSOS are refined by using this method.

Key words: secure operating system; code path; reference monitor; optimization; IPC redirection

高等级安全操作系统是国家信息安全保护的基础, 因而历来受到国际上的普遍关注. 以美国为例, 1985 年美国国防部发布了《可信计算机系统评测标准》(Trusted Computer System Evaluation Criteria, TCSEC), 它将计算机安全性分为 4 类 7 个级别[1], 确定了安全操作系统的开发标准. 美国军方更是使用 A1 级安全操作系统 ASOS 保护要害部门的信息安全[2]. 美国的大部分商业公司为了保护商业机密, 都广泛使用高等级安全操作系统, 例如波音公司就使用 A1 级安全操作系统 MLS-LAN 安全网络服务器[3]保护设计资料.

访问验证保护级安全操作系统是中华人民共和国国家标准《计算机信息系统安全保护等级划分准则》(GB17859-1999)[4]定义的目前最高等级的安全操作系统, 对中国国家信息安全等级保护战略以及中国基础

① 基金项目:中国科学院重大方向性项目(KGCX2-YW-125);国家自然科学基金(91218302,61432001)
收稿时间:2014-11-23;收到修改稿时间:2015-01-19

软件产业的提升和促进具有重要意义, 因此高等级安全操作系统在中国备受重视且需求巨大.

微内核架构很好地满足了高等级安全操作系统的要求. 微内核的设计思想是在内核中只保留必须在内核态运行的功能[5,6], 把其他功能都移至用户态以服务线程的方式实现, 使得微内核本身体积大大减小. 微内核中的权能机制是强制访问控制的一部分, 利用角色权能、进程权能和文件权能, 使得不同用户或同一用户分别执行相同或不同的文件都可以有不同的权限, 从而可以明确各个进程运行时的权限, 使各个进程仅具有完成其功能所必需的能力, 实现最小特权原则, 这一原则是安全操作系统最重要的特征之一. 微内核的主要功能是为上层任务提供执行的能力以及资源, 同时将上层任务强制分离, 并提供安全的通讯机制, 以便多任务能配合运行, 运行于微内核之上的用户态的系统服务处于各自的地址空间之内, 保证了服务之间的强隔离[6], 一个服务的崩溃不会影响到其他服务的正常工作, 因此基于微内核的操作系统具有很高的可重构性、稳定性和可靠性, 这些特性也都是高等级安全操作系统所必备的.

微内核操作系统在发展历程中的研究重点一直聚焦在性能的提高以及性能与安全的平衡. 安全与性能是一对矛盾, 矛盾的核心和焦点是 IPC(Inter-Process Communication, 进程间通信)[5]. 以 Mach 系统为代表[7,8]的第一代微内核操作系统满足了安全需求, 但性能不佳, 原因是微内核系统用独立进程来隔离系统服务, 服务间采用远程过程调用(Remote Procedure Call , RPC)进行通信[5], 并且频繁用于消息传输的 IPC 实现代码的空间局部性差, 使内核内部的命中率过低[6], 这些情况都导致了性能瓶颈, 造成系统整体性能大大降低[9]. 第二代微内核操作系统 L4 很大程度上解决了性能问题, 其设计以性能为第一目标[9], 许多关键的代码段都是用汇编语言写成[5,9], 以尽可能提高性能, 并使用 IPC 作为唯一的通信机制[5,9], 采用无缓冲同步 IPC[9]等多种实现方式使微内核中 IPC 性能获得大幅提高, 但 L4 过于追求性能, 严重违反了微内核设计原则如最小化原则, 因此安全大打折扣[5]. Minix3 是第二代微内核操作系统的另一个代表, 其将设备驱动程序、服务进程等完全的运行在用户空间[11], 性能上不如宏内核, 但提供了稳定安全的运行环境. 第三代微内核系统 Fiasco.OC 的设计和实现目标是在保障安全性的前提下追求更高的性能[12]. Fiasco.OC 由 L4/Fiasco 发展而来, 最显著的特征是其是基于权能(Capability)的系统[13]. Fiasco.OC 采用 C++作为开发语言, 合理使用 C++进行开发不会导致性能下降[5].

访问验证保护级安全操作系统的要求, 使得性能与安全的平衡问题更突出, 而面向访问验证保护级安全操作系统原型系统(VSOS)正是为了解决这一问题, 推进新一代高等级安全操作系统的研发. VSOS 除了关注内核是否完全可验证的问题以外, 仍然存在与 IPC 相关的性能问题, 这也是本文关注的重点.

本论文结构如下: 第 2 部分介绍 VSOS 的特点和架构; 第 3 部分介绍 VSOS 的核心安全组件——访问监控器的设计与满足的安全机制; 第 4 部分介绍针对访问监控器实现 VSOS 性能优化的方法; 第 5 部分介绍 VSOS 系统性能的测评方法和实验结果, 并利用测试方法验证系统运行的正确性和稳定性; 第 6 部分将对论文进行总结, 并进行了展望.

1 VSOS安全操作系统

GB17859-1999 第五级(访问验证保护级)是标准要求的最高等级安全操作系统, 是四级安全操作系统的向更高安全等级发展的结果. 此级别的操作系统要求[4]: 1)计算机信息系统可信计算基[14](Trusted Computing Base, TCB)满足访问监控器[15](Reference Monitor)需求, 访问监控器仲裁主体对客体的全部访问; 2)访问监控器本身是抗篡改的; 3)在设计和实现时, 必须足够小, 能够分析和测试, 从系统工程角度将其复杂性降低到最小程度. 从国内外安全操作系统研究和开发实践上看, 开发面向国标第五级“访问验证保护级”的操作系统原型系统 VSOS 具有极大的挑战性. 1972 年 James Anderson 报告提出了实现安全操作系统内核的四个方面的要求[15]: 1)不可旁过; 2)总是被调用; 3)尽可能小; 4)可验证. 然而, 不可旁过、总是被调用这样的要求会很明显的降低性能. 因为如果大量操作行为都要通过内核的处理, 将会导致内核成为性能的瓶颈, 整个系统变得不可用, 这是完全无法接受的.

VSOS 的设计和开发兼顾了安全和功能的需求, 基于支持微内核的 Genode 操作系统框架[16,17]进行开发, 在架构方面进行了创新, 并实现了多项安全机制, 满足了第五级安全操作系统的所有重要安全机制. 用户态程序之间的交互必须依赖于内核所提供的 IPC 服

务，基于 IPC 服务的特性添加相应的安全增强功能，相应的安全机制不可绕过. 由于内核态特权代码规模非常小，TCB 最小化的实现难度较小，而且传统宏内核内的非关键驱动等特定服务可以从 TCB 中移除.

面向访问验证级安全操作系统原型系统 VSOS 具有如下特点：1)VSOS 采用微内核作为基础，以层次化、模块化为方法解决了操作系统不可旁过、总是被调用的保障性问题；2)VSOS 支持安全管理员职能，扩充了审计机制，当发生与安全相关的事件时会发出信号，并提供了系统恢复机制；3)VSOS 满足的安全机制包括强制访问控制、标记、身份鉴别、审计、可信路径等十个方面的安全功能要求，这些安全功能的实现涉及了 VSOS 的各个层次，层次之间会产生大量的 IPC，因此也会对 VSOS 的性能产生一定影响，相关研究对象如图 1 所示.

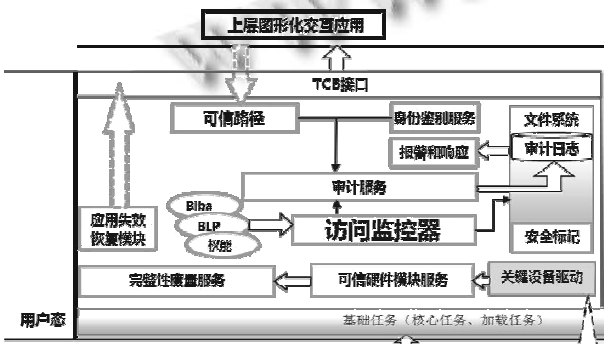

图 1　VSOS 架构中的研究对象

2　访问监控器的设计

访问监控器实现了用户身份的标识和鉴别、系统关键操作的安全审计以及基于权能的进程级主客体的强制访问控制.

微内核环境下，内核中只保留最小的功能，其它的模块都运行在应用层. 强制访问控制功能的实现，可以集中到访问监控器上. 微内核环境下，IPC 作为系统内各部分之间交互的唯一方式，处于系统交互中的关键位置，因此访问监控器建立在 IPC 之上. 通过限制 IPC 的信息流动，就可以实现强制访问控制. 根据 GB17859 要求[4]，应用的所有 IPC 访问请求都要经过访问监控器，访问监控器仲裁主体对客体的全部访问. 由于 IPC 作为系统内各部分之间交互的唯一方式，这种设计就很好的解决了访问监控器的不可旁过性.

访问监控器按照设定的规则限制 IPC 的信息流动，并结合在计算机资源上添加的标记，保护敏感数据的访问. 在访问监控器的基础上，除支持 BLP 保密性安全模型外，访问控制框架还支持多策略模型的扩展和多策略融合. VSOS 分别通过 BLP 模型和 Biba 模型来实现信息的机密性和完整性，各个模块都必须首先对系统的主客体分别赋予与其身份相对称的安全标记，即 BLP 安全标记和 Biba 安全标记，分别由级别部分和类别部分组成.

在 GB17859 第四级中，只要求提供用户初始登陆与鉴别过程的可信路径，但在第五级中的要求是[4]：“提供可信计算基与连接用户之间的可信通信路径，路径只能由可信计算基和该用户激活，路径上与其他路径上的通讯相隔离，且能正常区分.” 如图 2 所示，可信路径机制为工作在系统本地虚拟终端上的用户提供在本地系统上登录与鉴别过程中的可信路径，可信路径只能由用户通过安全注意键[18](Secure Attention Key，SAK)机制激活. 用户在键盘上进行 SAK 击键操作，操作通过系统键盘中断程序传送到当前终端. 终端会执行身份鉴别应用，架构如图 3 所示，系统初始登录与鉴别进程通过某一虚拟终端与用户进行交互，进程通过虚拟终端的输出提示用户输入登录与鉴别信息，用户输入信息后进程读取虚拟终端的输入获取信息，进行必要的检查，并将结果通过虚拟终端返回给用户. 系统的显示器上会显示当前终端用户虚拟屏幕的内容，用户需要查看非当前用户虚拟屏幕的内容时，可以通过 SAK 键切换.

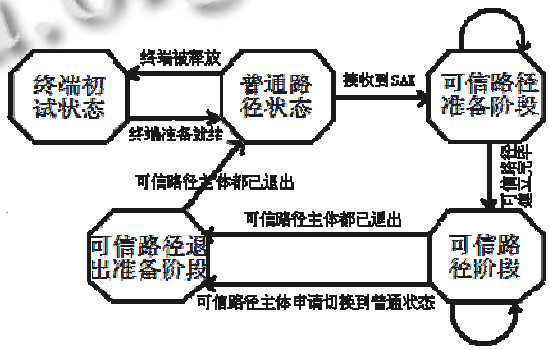

图 2　虚拟终端设备的可信路径状态

GB17859 第五级审计机制在第四级的基础上，提出了更高要求[4]：“提供安全管理员职能，提供监控可审计事件发生和积累的机制，在特定的条件下向安全管理员报警，甚至中止这些事件”. 针对以上要求，本

项研究将基于安全体系结构的研究中确定的安全相关事件, 并通过安全服务器为各种安全机制提供的接口, 设计实现针对模板的审计记录自动分析工具, 以及修改系统安全策略的机制.

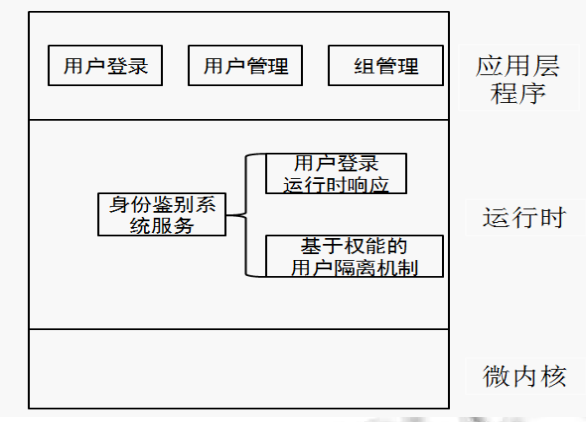

图 3 身份鉴别架构图

3 针对访问监控器的性能优化

访问监控器作为内核基础服务, 满足不可旁过性和总是被调用, 所以所有上层应用程序都要经过访问监控器仲裁和判断, 而如果在仲裁过程中进行时间消耗较大的操作, 将会在很大程度上影响系统的性能和可用性.

基于 IPC 系统调用之上实现的访问监控器, 负责仲裁系统内权能的授予, 验证通信对象已有权能的正确性, 并能按照既定的安全策略仲裁用户的 IPC 请求. 权能是一个二元组, 包含所指向的实体和实体所具有的相应权限. VSOS 的访问监控器采用基于 IPC 重定向[19]的方式为原则进行实现, 对应到五级标准要求里的访问监控器的不可旁过性.

具体实现访问监控器时, 如图 4 所示. 应用 T1 请求和 T2 进行交互的时候, 需要创建内核对象 IPC Gate. 将 IPC Gate 的权能 cap 授权给 T1 和 T2, 它们就可以使用 IPC 请求进行通信. 要实现应用间的访问控制, 必须先阻断应用 T1 的 IPC 请求, 使其经过访问监控器的仲裁, 实现 IPC 重定向.

IPC 重定向的具体过程是应用 T1 和应用 T2 经过访问监控器的仲裁后建立新的IPC, 步骤如下: 1)应用T1通过 IPC Gate 与访问监控器通信; 2)访问监控器通过授权模块和安全策略库决定应用 T1 是否可以与 T2 通信; 3)访问监控器通过IPC Gate 建立与应用T2的通信, 即将访问监控器嵌入到应用之间, 起到中转通信作用.

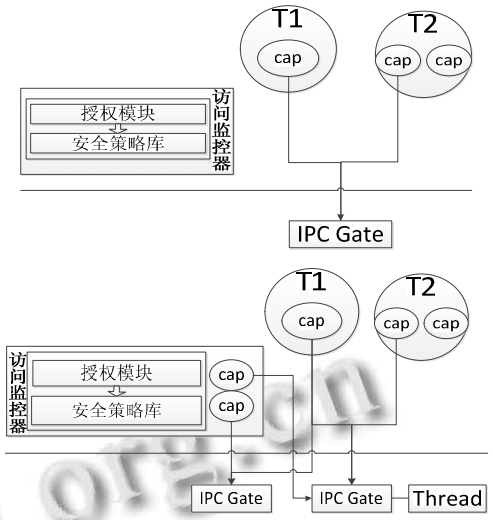

图 4 应用 T1 与 T2 的 IPC 重定向通信示意

下面以 VSOS 中具体应用程序的 IPC 过程为例, 分析和描述IPC重定向中的信息流. 如图5所示, 所有得到执行的 RPC 操作都要执行此段代码, 判断是否是读、写或者是同时读写的状态, 如果 RPC 操作是上述三种状态, 则调用 monitor_judge 方法进行仲裁. 根据当前的 RPC 操作类型(fun_name)和操作码(op, 判断当前的RPC操作是否为空操作void), 访问监控器首先通过 monitor_judge 方法分别对主体 T1 和客体 T2 的 ID 进行仲裁, monitor_judge 依据 BLP 或 Biba 模型对主体和客体 ID 各自对应权能中记录的安全级别进行比较, 若符合 BLP 或 Biba 模型要求, 则 T2 可通过访问监控器得到T1传递的信息; 若不符合, 则拒绝此RPC操作并返回 RPC 拒绝消息, 并通过上层应用将操作失败信息返回给用户. 根据需要, VSOS 中可以使用 BLP 或者 Biba 的一种, 在安全级别管理应用中安全员还可以设置各个用户的 BLP 或 Biba 等级.

```
Rpc_exception_code exc;
{
        ......
        ......
        static Monitor_Connection h;
        if(strcmp(fun_name,"read_state")==0||
        strcmp(fun_name,"write_state")==0||
        strcmp(fun_name,"change_state")==0){
        if(h.monitor_judge(senderID,receiverID,fun_name,op)==0){
            return RPC_PERMISSION_DENIED;
        }
    }
}
```

图 5 访问监控器在 RPC 服务器中进行仲裁

IPC 是进程交互的主要实现手段. 系统设计中, 所有的用户态任务将以服务的形式运行, 任务之间以 IPC 的方式进行交互. 例如内存分配时, Sigma0 拥有所有的内存, 通过 Map 和 Grant IPC 将内存分配给其他任务. 页面管理时, 每个线程对应一个单独的 Pager 线程[9], 负责处理相应的缺页请求, 线程、内核和 Pager 之间通过 IPC 进行交互. 文件访问时, 向磁盘驱动发送 IPC 请求等. IPC 的可控性是访问监控器设计的重要依据.

因为微内核架构下所有的通信都经过 IPC, 通过 IPC 重定向建立的通信机制可以使访问监控器仲裁应用间的通信请求, 根据请求内容的不同, 采用相应的处理方式. 如果需要申请获得权能, 或者访问对象带有敏感标记, 访问监控器会根据安全策略库中的内容对请求进行仲裁.

基于 IPC 重定向原理实现访问监控器为 VSOS 系统带来诸多安全特性, 例如: 1)安全功能结构化: 实现更强的隔离性, 所有交互(同一用户不同应用之间以及不同用户应用之间)都是初始化赋予的或者经过访问控制策略授权; 2)可信路径: 通过控制 IPC 的可信, 确保终端与 TCB 之间的可信连接; 3)访问监控器不可旁过: 通过访问监控器进行 IPC 是应用间通信的前提.

但是, 由于访问监控器在多个应用程序交互、安全机制满足以及 IPC 拦截过程中都要进行仲裁, 如果其实现代码中有性能消耗严重的操作, 会造成系统性能极其低下, 用户在操作过程中会明显感到系统很缓慢, 几乎不可用. 在实际开发中, 这种情况多次出现, 而引起这种现象的原因并不完全相同, 但都对 IPC 重定向方式的访问监控器产生了不良的性能影响. 本论文使用的方法是基于调整代码执行路径进行优化的, 从两个不同方面实现了 VSOS 的性能优化.

3.1 访问监控器调用方式与优化

访问监控器在内核组件、服务以及上层应用等多个地方被调用. 对于上层应用, 应用程序启动面板调用访问监控器, 用来仲裁应用的运行权限、当前用户对此应用的运行权限以及此应用对用户的可见性(权限不符合要求的用户无法访问特定应用的运行入口, 如启动按钮)等. 访问监控器仲裁上层应用是通过 C/S 模式完成的. 应用程序启动面板作为客户端, 与访问监控器建立连接, 调用相关仲裁方法, 根据 BLP 模型或 Biba 模型来仲裁当前特定安全级别的用户对某特定应用有何种权限. BLP 模型的基本安全策略是下读上写, 即高安全级的主体只能读低安全级的客体, 低安全级的主体只能写高安全级的客体, 这保证了信息只能从低向高的安全级流动, 使敏感信息不会泄露; 而 Biba 模型则反之.

应用程序启动面板、上层应用等通过远程过程调用(Remote procedure call, RPC)方式来调用访问监控器中的仲裁方法进行仲裁、判断等操作, 而客户端向服务器读写数据时, 则是直接在 RPC 服务器中进行 IPC 重定向, 如图 5 所示.

图 5 所示的代码存在环形调用问题, 即在执行 RPC 操作时与访问监控器(Monitor)服务建立连接, 调用其相应方法进行仲裁, 但建立连接就需要大量的 RPC 操作, 这样就会有对系统的整体性能产生影响的风险, 但如果仲裁方法 monitor_judge 执行效率高, 不包含大量耗时的操作, 则不会造成使用者可感知的性能下降. 本系统 VSOS 在之前遇到了性能问题, 原因之一就是 monitor_judge 中调用了执行效率很低的实时时钟组件(Real-Time Clock, RTC)用于生成审计日志所需的日期和时间, 导致在执行上层应用读写操作时, 系统反应迟缓, 使用者的体验很差. 在排除了访问监控器其他部分对性能的影响后, 确定了上述组件的使用是导致读写操作效率降低的原因, 因此采用了 RTC 和时间戳读取组件(Read Time Stamp Counter, RDTSC)结合的方法生成日期和时间.

首先在系统启动时, 利用一个与使用者交互较少的上层应用(例如版本号显示应用完全不需要与用户交互)来生成 RTC 时间, 并记录一个时间戳 RDTSC1. 在需要生成时间的程序中, 再次记录时间戳 RDTSC2, 最终得到的时间 Time 如公式(1)所示:

$$Time = RTC + (RDTSC2 - RDTSC1)/CPU\ 频率 \quad (1)$$

公式(1)中 Time 的单位是秒(s), CPU 频率的单位是赫兹(Hz). 这种方式所得到的时间足够精确, 在利用当前计算机所使用的 CPU 频率进行校正后, RDTSC 与真实时间的误差不大于 50 微秒, 而这里用于显示日期和时间, 精度只需要达到秒级即可.

在目前 Fiasco.OC 中没有很好的时间获取组件的情况下, 这一实现方式是一种比较好的解决方案. 此方案存在的缺点是系统启动时仍然需要调用 RTC 来获取日期时间, 并且在需要记录日期时间时, 需要使用

公式(1)进行计算才能得到时间, 以上操作会消耗一些时间, 并且可能存在误差. 预期解决方案是利用 RDTSC 获取时间戳, 然后自行实现将时间戳转换成日期时间的转换函数, 这样可以避免使用 RTC 等效率较低的日期时间获取组件, 并且可以减少误差.

3.2 可信路径机制优化

可信路径机制的执行过程从按下 SAK 键一直贯穿到应用程序启动面板启动, 直到再次按下 SAK 键, 又进入可信路径过程. 可信路径机制提供可信的图形化登录视图, 其他客户端无法获取登录视图内的输入, 能满足身份认证与鉴别机制的要求, 另外还为访问监控器进行策略决策提供可视化的依据.

对于可信路径机制的优化, 重点主要集中在身份验证应用、从应用程序启动面板再次进入可信路径过程、以及从应用程序启动面板上启动应用程序的过程.

身份验证应用出现的问题是在输入用户名和密码后, 点击登录按钮后, 无论用户名密码正确与否, 都会出现短暂的假死现象. 经过测试, 发现出现这种现象的原因是与访问监控器建立连接, 以及写入审计日志的代码, 都集中在身份鉴别事务处理完毕后和将消息返回到用户终端之间, 这导致此阶段的时间消耗增加, 因此用户体验会受到很大影响. 对于这种情况, 利用分散的方法将耗时长但又不是必须在当前阶段执行的代码分别向前或向后调整, 这样程序的执行逻辑没有变化, 但是用户体验增强了.

在已启动应用程序启动面板的情况下, 按下 SAK 键, 启动面板消失, 身份验证应用重新启动, 再次进入可信路径过程, 这一过程中对耗时最多、对性能影响最大的阶段就是自动关闭所有已打开应用的过程, 关系程序释放资源的过程非常耗时. 关闭所有应用程序的过程如下: 首先利用 QT 控件的一个方法 findChildren 来查找所有已启动的子进程对应的 entry(child_entry), 存入一个 List 中, 然后利用 for 循环逐个关闭 List 中的 child_entry 并释放它们占用的资源. 解决这一问题的方案是采用分散的方法来调整优化代码路径, 并且为了更好的用户体验, 可以先将每一个子进程对应的窗口隐藏, 然后在后台进行关闭操作, 这样也能增加用户的体验. 在释放子进程资源的过程中, 访问监控器也会发挥作用, 因此使用者会很明显地感到系统响应慢, 但是通过 3.1 节对访问监控器的优化, 此处的性能也有显著提高.

从应用程序启动面板启动应用程序的过程中需要访问监控器参与对应用的仲裁, 如果当前用户的权限不能启动特定应用, 则访问监控器会通过 IPC 重定向的方式拒绝 RPC 操作, 返回一个禁止信号, 上层应用弹出对话框提示用户没有权限. 通过 3.1 节对访问监控器的优化, 此处的性能也大大提高, 达到了流畅无延迟打开应用程序的效果. 在此处通过调整代码路径, 也可以一定程度上提高性能. 因为无论用户的权限是否足够打开应用, 访问监控器都会进行仲裁并记录到日志中, 因此可以等到事务处理代码执行结束后再执行写日志操作, 通过一个 bool 变量可以记录用户是否成功启动应用, 然后理由条件判断语句来执行响应的写日志操作.

4 性能测评

本实验对 VSOS 优化前后的若干主要上层应用和操作进行运行时间测试, 并进行对比. 实验采用的是计时器中的 Timer::Session::elapsed_ms 方法, 其精度为毫秒级, 满足本实验的要求, 而且此方法是针对 Fiasco.OC 内核实现的, 其他内核中没有实现此方法, 因此可以更好的进行时间测量, 提高时间测量的准确性. 通过 10 次测量取平均值, 得到 VSOS 优化前后代表性上层应用的各自运行启动时间对比, 如表 1 所示.

表 1 代表性上层应用的启动运行时间

启动耗时(单位:s) 是否优化 / 应用名	优化前	优化后
应用启动面板	0.290	0.030
中密级应用	0.016	0.002
中密级服务	0.016	0.002
读操作	0.071	0.004
写操作	0.074	0.005
读/写操作	0.140	0.008

由图 6 所示, 优化后的运行速度提升了约 10 倍. 读、写、读/写操作都是在启动程序后首次执行相应操作时测量得到, 这时中密级应用程序先要与中密级服务程序建立连接, 然后再进行相应操作. 如果不是初次执行某特定操作, 由于已经实现建立好连接, 执行此操作的耗时会更短. 此外, 访问监控器如果效率低, 对 QT 应用程序的界面显示速度也有很大影响, 虽然优化前上层应用的运行时间均没有超过 0.5 秒, 但界

面绘制和显示效率低，用户体验也会很差，因此实际的运行时间还要增加 1-2 秒. 访问监控器经过优化后，程序事务处理代码的执行和界面显示速度都得到了提高，因此实际的运行时间与测量得到的时间基本相同.

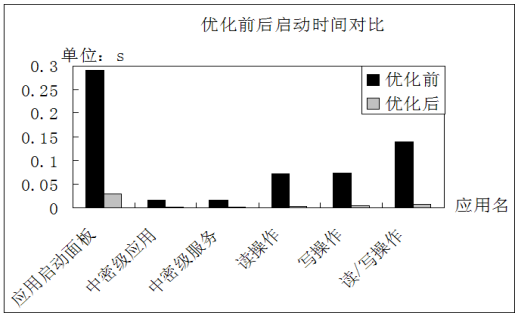

图 6 优化前后代表性上层应用启动时间对比

5 结语

本文主要做了如下贡献: 介绍了安全操作系统原型系统 VSOS 的结构和特点，设计和实现了 VSOS 的安全核心组件访问监控器，满足了安全操作系统安全内核的不可旁过和总是被调用的要求; 利用基于代码路径的方法改进了访问监控器的调用方式，使 VSOS 的性能得到了提升; 对可信路径安全机制的实现方式进行了代码路径优化，使可信路径过程的用户体验得到了提升，使 VSOS 操作系统兼具安全性、功能性和可用性. 此外，本文还提出了一种新的记录日期时间的方法，由于显示时间组件自身的缺陷，方案并不完美，但也为解决此问题提供了一个可用的方案.

本文所介绍的系统 VSOS 在性能优化方面还有大量创新点可以进行研究，除了上面提到的时间获取问题，访问监控器的实现架构也是非常值得研究的主题.

参考文献

1 US Department of Defense. DoD 5200.28-STD: Trusted Computer System Evaluation Criteria (Orange Book). DoD Rainbow series, 1985.

2 Waldhart NA. The army secure operating system. Proc. of the 1990 IEEE Symposium on Research in Security and Privacy. Oakland, CA. IEEE Computer Society Press. 1990. 50–60.

3 Loscocco PA, Kutz WR, Johnson DM, Watro RJ. Dealing with the dynamics of security: Flexibility with utility in an MLS LAN. Proc. of the Eighth Annual Computer Security Applications Conference. San Antonio, Texas. IEEE Computer Society Press. 1992. 180–192.

4 中国国家质量技术监督局.中华人民共和国国家标准: 计算机信息系统安全保护等级划分准则.GB17859-1999, 1999-09-13.

5 Elphinstone K, Heiser G. From L3 to seL4: What have we learnt in 20 years of L4 microkernels? Proc. of the 24th ACM Symposium on Operating Systems Principles. New York. ACM Press. 2013. 133–150.

6 Liedtke J. On μ-Kernel construction. Proc. of the 15th ACM Symposium on Operating Systems Principles. New York. ACM Press. 1995. 237–250.

7 Härtig H, Hohmuth M, Liedtke J, Schönberg S, Wolter J. The performance of μ-kernel-based systems. ACM SIGOPS Operating Systems Review, 1997, 31(5): 66–77.

8 Golub D, Dean R, Forin A, Rashid R. Unix as an application program. Proc. of the 1990 Summer USENIX Conference. Anaheim, CA. USENIX Press. 1990. 87–95.

9 Liedtke J. Improving IPC by kernel design. Proc. of the 14th ACM Symposium on Operating Systems Principles, Asheville, NC. ACM Press. 1993. 175–188.

10 Other system calls. Fiasco/L4 System Call C-Bindings Reference Manual. http://os.inf.tu-dresden.de/l4env/doc/html/l4sys-l4v2/group__api__calls__other.html.

11 Tanenbaum AS, Woodhull AS. Operating systems: Design and implementation. 3rd ed, Upper Saddle River, NJ: Prentice Hall, 2006.

12 Fiasco. OC webiste. https://os.inf.tu-dresden.de/fiasco/.

13 Levy HM. Capability-Based Computer Systems. Digital Press. 1984. ISBN 0-932376-22-3.

14 Rushby J. Design and verification of secure systems. Proc. of the 8th ACM Symposium on Operating System Principles. Pacific Grove, CA. ACM Press. 1981. 12–21.

15 Anderson JP. Computer Security Technology Planning Study. Vol.1:Hanscom AFB, Bedford, MA, ESD-TR-73-51, Electronic Systems Division, Air Force Systems Command. 1973. 22–23.

16 Feske N, Helmuth C. Design of the Bastei OS Architecture. Vol.1: Dresden, Germany, TUD-FI06-07, TU Dresden, 2006.

17 Genode webiste. http://genode.org.

18 Morton A. Linux 2.4.2 Secure Attention Key (SAK) Handling. Linux Kernel Organization. 2001-03-18.

19 Jaeger T, Elphinstone K, Liedtke J, Panteleenko V. Flexible access control using IPC redirection. Proc. of the 7th Workshop on Hot Topics in Operating Systems. Rio Rico, AZ. ACM Press. 1999. 191–196.