*

Space 文件系统安全可靠机制的研究与实现

刘金刚[1,2]，　李　宁[1,2]
(1. 首都师范大学　计算机科学联合研究院,北京　100037;2. 中国科学院　计算技术研究所,北京　100080)

摘　要:首先通过对传统文件系统在安全可靠方面存在的问题进行研究分析,然后提出了一种基于 Linux 操作系统的文件系统安全可靠机制,即将操作系统安装在相对独立的物理空间中,与其他应用软件在物理空间上分隔开来,同时将安装操作系统的物理存储介质限定为只读方式,使所安装的操作系统在物理层上不可被改写,从而达到安全可靠;同时将用户空间在物理上与系统空间相互隔绝,当用户需要安装软件或保存各种文档时,可以透明地存储在该空间中。透明是指对用户来说,看到的文件系统视图或在操作方式上与传统操作系统完全相同,感觉不到底层系统空间和用户空间的存在。同时,详细描述了提出的 SpaceFS 文件系统的设计原理和实现细节,最后通过对其进行充分的测试数据,得出了在不损失系统性能的基础上保证了安全可靠的结论。

关键词:Linux;Space 文件系统;安全可靠
中图分类号: TP316　　**文献标识码**:A　　**文章编号**:1009 - 3516(2008)06 - 0080 - 05

对于传统操作系统,用户的误操作、外来的恶意攻击或病毒极有可能造成系统崩溃,而现有的权限管理、访问控制、安全审计等方法对拥有超级权限的用户几乎无能为力;另外,由于需要在用户行为上施加各种策略,所以对系统的性能有一定影响。文中所设计并实现的 SpaceFS 通过在虚拟文件系统和本地文件之间添加一个逻辑层,在不影响用户正常使用的前提下提供了一个文件系统安全可靠机制,并从系统二义性[1]和压栈式操作[2]2 个方面详细描述了其设计原理和实现细节。

1　设计

1.1　Linux 文件系统工作机制

文件系统作为整个操作系统最重要的子系统之一,按功能划分为 2 类:虚拟文件系统[3](Virtual File-system, VFS)和本地文件系统(如 Ext3 和 Fat32)。VFS 相当于用户的应用程序与本地文件系统实现之间的一个抽象层,用来处理与 Linux 标准文件系统相关的所有系统调用[4];本地文件系统则是用于组织和管理存储于磁盘、网络等存储介质上的数据的一组方法和数据结构。二者之间具有如图 1 所示的调用关系,现假定用户欲读取某位于 Ext3 磁盘分区上一个文件的内容,当用户态进程发起一个 read 系统调用时,该系统调用的服务例程 sys_read 会依次调用 VFS 层的

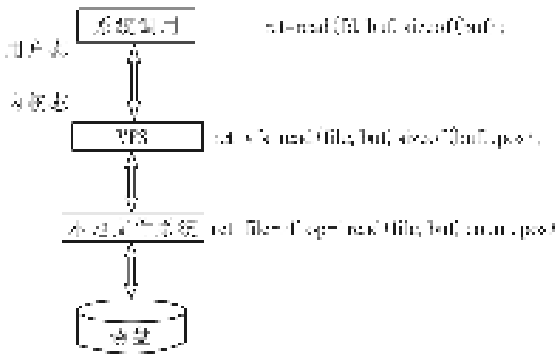

图 1　VFS 与本地文件系统之间的调用关系
Fig. 1　Call sequence between VFS and native file system

* **收稿日期**:2008 - 04 - 09
作者简介:刘金刚(1963 -),男,辽宁铁岭人,教授,博士生导师,主要从事智能接口技术方面的研究.
E - mail:jason. li@ sun. com

函数 vfs_read,在后文会看到,被打开的文件在内核内存中是由 1 个 file 结构体来表示的,该结构体中包含 1 个称为 f_op 的字段,该字段中包含一组指向专对特定本地文件系统(如本例中的 Ext3 文件系统)的函数指针,当然还包括读文件的函数,vfs_read 查找到指向该函数的指针,并调用它。这样一来,应用程序的 read( ) 系统调用就被转化为相对间接地底层调用 file - >f_op - >read( )。

1.2　SpaceFS 的设计思想

虽然 Linux 支持许多不同种类型的文件系统,如基于磁盘的文件或基于网络的文件系统等等,然而开发出一个稳定且高效的文件系统需要花费开发人员数年的心血,并且一旦能够稳定且高效的工作之后,通过对其进行修改来增加新的功能往往是得不偿失的。另外内核文件子系统是一个非常复杂的运行环境,一个非常小的错误甚至能够导致文件数据的丢失或者系统的不稳定,所以也就不难理解诸如 Ext2、Fat32 这样被广泛使用的文件系统为何数年没有更新了。SpaceFS 在不修改虚拟文件系统和底层本地文件系统的情况下采用了一种全新的设计模式,不仅便于高效地开发调试,而且在性能方面不会有明显的损失。主要设计思想如图 2 所示,SpaceFS 相当于虚拟文件系统与本地文件系统之间的一个逻辑层。

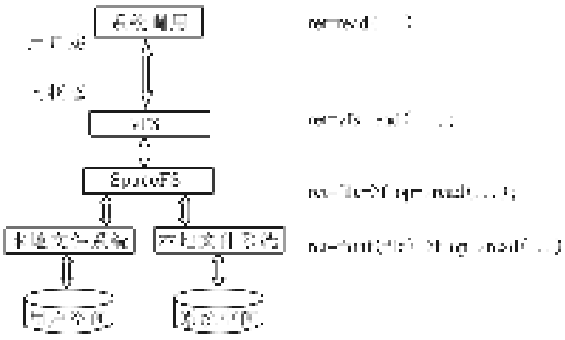

图 2　SpaceFS 设计框架

Fig. 2　Design framework for SpaceFS

从虚拟文件系统的角度看,其相当于一个本地文件系统;而从本地文件系统的角度看,其又相当于虚拟文件系统。系统空间内存放一个完整的 Linux 操作系统,用户空间存放系统空间的改动副本(如更改的配置文件或安装的软件)以及用户的各种文档,2 个空间分别位于不同的物理区间中并且可能具有不同的文件系统类型(如系统空间可安装至 Ext3 分区上而用户空间安装至 ntfs 分区上)。在系统启动时,SpaceFS 合并系统空间和用户空间的文件系统树,并被添加到虚拟文件系统树中,所以从用户的角度看,整个系统如同存储在文件系统类型为 SpaceFS 的单一分区中一样。由于系统空间始终处于原始状态,所以当系统无法正常使用甚至崩溃时,只需对用户空间的某一部分进行处理即可,从而保证了整个系统的安全可靠。为了实现如上所述的功能,SpaceFS 需要重点解决 3 个方面的问题。

第 1 个问题是在系统空间和用户空间的同一目录位置可能存放有相同文件名但是内容不同的 2 个文件。例如当用户对系统空间的/etc/hosts 文件进行修改时,其修改后的副本就会保存在用户空间的同一目录位置,而用户通常期望看到修改后的版本,否则会导致许多程序崩溃。解决该问题的方法是采用优先级机制,赋予用户空间相对系统空间较高的优先级,如果 2 个空间中的同一目录位置出现具有相同文件名的文件,则只选择用户空间中的文件版本。

第 2 个问题是文件的删除。由于在 2 个空间中可能出现具有相同文件名的文件,并且系统空间不允许被修改,所以文件的删除需要小心地处理,否则会引起混乱,因为一个在用户看来已经被删除的文件还继续存在。解决该问题的方法是,当被删除的文件存在于用户空间中时,直接将其删除,并检查系统空间中同一目录位置是否存在具有相同文件名的文件,如果存在,则在用户空间的同一目录位置创建一个不包含数据的空文件用于遮挡系统空间中的相应文件,例如如果被删除的文件名为 F,则可创建文件名为_zd_F 的遮挡文件。当被删除的文件存在于系统空间中时,同样在用户空间的同一目录位置创建一个遮挡文件。当进行查找操作时,如遇到文件名为_zd_F 的文件,则会忽略系统空间的相应文件。

第 3 个问题是如何保持各种操作的原子性。由于一个 SpaceFS 操作会包含 2 个甚至多个底层本地文件系统操作,所以维护其原子性显得格外重要,操作结果只能是成功或者失败,不能出现部分成功的情况,否则系统会处于一个不一致的状态。解决该问题的方法是小心选择操作的顺序,例如在删除操作中,2 个空间中都有文件名为 F 的文件,如果先删除用户空间中文件 F,再创建遮挡文件_zd_F,若在 2 个操作之间机器突然断电,就会出现混乱,因为用户会看到系统空间中的文件 F;相反如果先创建遮挡文件_zd_F,再删除用户空间中文件 F,就不会出现任何问题。

2 实现

2.1 Linux 通用文件模型

通用文件模型[5]由下列 4 种对象类型组成:①超级块对象(superblock)用于存放已安装文件系统的有关信息,如挂载选项以及文件系统类型。超级块操作包括读取文件系统统计信息、分配索引节点等;②索引节点对象(inode)用于存放关于具体文件的数据和属性信息,如所有者、访问权限和文件大小。索引节点操作包括创建、查找和删除等;③文件对象(file)用于存放打开文件与进程之间进行交互的有关信息,每一个用户态文件描述符对应一个文件对象。文件操作包括打开、读和写等;④目录项对象用于存放文件的特定名称与对应文件进行链接的有关信息。目录项操作包括生成哈希值和比较 2 个文件名等。如图 3 所示是一个简单的示例,说明进程以及各种对象类型之间是如何交互的,连接线表示的是前端对象的地址保存在末端对象的某一个字段里,如 dentry 对象的地址保存在 file 对象的 f_dentry 字段中。在图中,3 个不同进程已经打开了同一个文件,其中 2 个进程使用使用同一个硬链接。在这种情况下,其中的每个进程都使用自己的文件对象,但只需要 2 个目录项对象,每个硬链接对应一个目录项对象。这 2 个目录项对象指向同一个索引节点对象,该索引节点对象标识超级块对象,以及相应的磁盘上该文件的数据块。

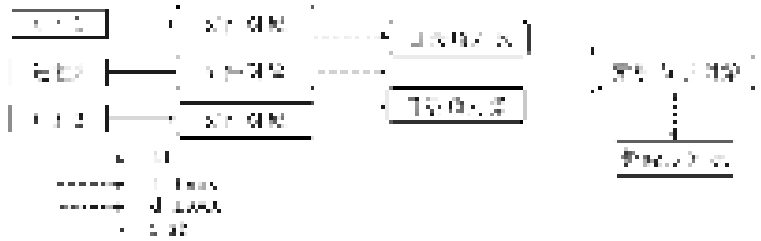

图 3 进程以及各种 VFS 对象之间的交互

Fig. 3 Interactivity between processes and VFS objects

2.2 SpaceFS 的初始化

在系统引导阶段,SpaceFS 被初始化并添加到 VFS 树中作为根文件系统,在这一过程中,主要完成两个任务:①分配 SpaceFS 超级块对象(superblock)并将其初始化,包括属性字段、操作以及指向底层本地文件系统超级块对象的指针等。②分配 SpaceFS 根目录的索引节点对象(inode)和目录项对象(dentry)并将其初始化,包括属性字段、操作以及指向底层本地文件系统根目录的索引节点对象和目录项对象的指针等。SpaceFS 在初始化阶段,像所有其它文件系统一样,并没有在内存中生成一棵完整的目录树,因为这是不必要的而且浪费内存并拖延系统启动速度[6]。SpaceFS 与系统空间和用户空间已经建立起了一个层次结构,在系统运行过程中,会根据需要动态地逐步地生成完整的树,如当用户第 1 次访问/usr/src 目录时,会动态地创建对应于 usr 和 src 的各种 VFS 对象并添加到 SpaceFS 目录树中。

2.3 SpaceFS 的实现细节

SpaceFS 定义了所有 VFS 对象类型的操作[7],其中一部分仅仅将 VFS 层的调用及参数传递给底层本地文件系统,如所有的文件操作,因为某一文件在某一时刻只能有一个副本被打开,要么是系统空间的副本,要么是用户空间的副本;而其他所有操作都会小心地以一定的顺序调用 2 个底层本地文件系统对象的方法。下面将详细分析其中几个较为关键的操作:

1)查找(lookup)

查找是最重要的索引节点操作之一[8]。它以一个目录的索引节点(inode)和该目录下的一个目录项(dentry)作为参数,并返回该目录项对应的索引节点。如果没有找到,将返回一个负状态的目录项(即为对应索引节点已不复存在的目录项)。SpaceFS 以优先级从高到低的顺序进行查找(即先用户空间后系统空间)。如果在用户空间找到并且是一个文件,则终止并返回相应的索引节点;而如果在用户空间找到并且是一个目录,则继续查找系统空间,并且如果同样找到一个目录,则与前者一起作为结果返回。这种情况发生在当用户读取一个目录(如/usr)时,需要将两个空间相应目录的内容合并在一起返回给用户。而如果在系统空间找到一个文件时,则被忽略,因为用户空间的优先级高。当在用户空间没有找到时,则查找系统空间并返回查找结果。

2)读目录(readdir)

读目录操作返回一个被打开目录下所有目录项对象,在 SpaceFS 中,这一过程是由系统空间的读目录操作与用户空间的读目录操作共同完成的,按照优先级从高到低操作,即先读取用户空间的目录。如果具有相同文件名的文件或目录同时出现在两个空间中时,则忽略系统空间中的版本。如果用户空间出现名为_zd_F 的遮挡文件,则忽略文件 F。为了达到以上目的,需要将已读取的文件或目录记录在一个哈希表中(使用文件名生成哈希值)。

3)创建文件

用于创建文件的操作包括 create、mkdir、symlink、mknod 和 link 等,虽然这些操作创建不同类型的文件,但是其实现细节基本相同。SpaceFS 使用查找操作返回的负状态的目录项对象创建新文件。然而,用户空间中如果存在被查找文件的遮挡文件,则需要分情况作额外处理:①创建文件的类型为普通文件,则将_zd_F 重命名为 F 并返回;②类型为目录、设备文件或符号链接,则创建新文件并删除遮挡文件。要特别注意以下情况,如系统空间中存在目录/a/b/c,然后在用户空间创建子目录 b 的遮挡文件,这时如果再重新创建子目录 b,则创建成功后不应该出现子目录 c,因为在用户看来其已经被删除了,所以需要在新创建的 b 目录下创建 c 的遮挡文件,这样才不会令文件系统发生混乱。

4)重命名文件(rename)

在任何文件系统中,重命名都是最复杂的操作之一,而在 SpaceFS 中它变得更加复杂,因为可能涉及到 2 个底层本地文件系统的重命名操作,而这期间又可能调用如 create、unlink、read 和 write 这样的操作。以下分别就几种情况进行讨论:①被重命名的文件只在用户空间存在:只需调用用户空间所在文件系统的重命名操作即可。②被重命名的文件只在系统空间中存在:首先将其复制到用户空间并创建可能存在父目录,然后调用用户空间所在文件系统的重命名操作。③被重命名的文件同时存在于两个空间:首先在用户空间创建遮挡文件来遮挡系统空间的文件版本,然后调用用户空间所在文件系统的重命名操作。

3　性能分析

本文所采取的性能评估方法为:通过在传统的 Linux 操作系统和安装了 SpaceFS 的 Linux 操作系统上运行具有代表性的 IO 密集型应用[9],来比较其性能差异。测试机器的配置为 1.4 GHz Pentium M 处理器、256 MB 内存以及 80 G IDE 硬盘,所采用的 Linux 操作系统为 Fedora Core 3 并且安装了所有截止到 2005 年 5 月 17 日的更新,所采用的底层本地文件系统分别为 Ext3 和 NTFS,因为它们已经得到广泛的应用并且非常稳定。测试的第 1 个应用为编译软件包,第 2 个应用为编写针对文件系统的压力测试脚本,2 种测试都会执行大量的各种 IO 操作。为了使得测试结果更加全面,针对不同规模也进行了测试。由于只关心 SpaceFS 的整体性能,所以采用的测试基准为时钟时间[10],而不单独考虑系统 CPU 时间及用户 CPU 时间。测试方法为编写 C 测试程序来得到每一测试运行的时间。

表 1　SpaceFS 性能测试对比

Tab. 1　Performance contrast for SpaceFS

	规模	传统的 Linux 操作系统	安装了 SpaceFS 的 Linux 操作系统
编译 openssh 4.0	155 个目标文件、1 个库文件、11 个二进制文件以及 4 个脚本文件	1 min33 s	1 min35 s
编译 2.6.18.1 版本内核默认配置下的所有模块	5 468 个目标文件	43 min52 s	44 min39 s
压力脚本测试 1	创建 5 级 5 叉树目录、在各目录下拷贝大小为 10K 的文件	28 s	28 s
压力脚本测试 2	创建 7 级 6 叉树目录、在各目录下拷贝大小为 1K 的文件	43 min37 s	47 min48 s

从以上结果可以看出,安装了 SpaceFS 的 Linux 操作系统并没有明显的性能损失。也就是说,SpaceFS 既能透明地提供安全可靠机制,又不会给用户的使用带来不便。

4 结论

本文设计并实现一种安全可靠的 Linux 文件系统,并且其性能相对传统操作系统只有很小一点损失。相比于其它设计方案,其无论在开发效率还是可维护性及可扩展性方面都有很大的优势。利用这种设计模式,开发人员可以很容易地扩展其功能,如加入另一个逻辑层以提供对用户空间的加密/解密或者备份功能。

参考文献:

[1] Loscocco P A, Smalley S D. Meeting Critical Security Objectives with Security - Enhanced Linux [C]//Proceedings of the 2001 Ottawa Linux Symposium, Ottawa: [s. n.], 2001.

[2] Anderson D, Chase J, Vadhat A. Interposed Request Routing for Scalable Network Storage[C]// In Proceedings of the 4th Usenix Symposium on Operating System Design and Implementation San Diego, CA: USENIX Association, 2000: 259 - 272.

[3] Ghormley D P, Petrou D, Rodrigues S H, et al. SLIC: An Extensibility System for Commodity Operating Systems[C]// In Proceedings of the Annual USENIX Technical Conference, Berkeley, CA: ACM, 1998: 39 - 52.

[4] Jones M B. Interposition Agents: Transparently Interposing User Code at the System Interface[C]//In Proceedings of the 14th Symposium on Operating Systems Principles (SOSP 93), Asheville, NC: ACM, 1993: 80 - 93.

[5] Korn D G, Krell E. A New Dimension for the Unix File System[C]//Software: Practice and Experience, New York, USA: John Wiley & Sons Inc, 1990: 19 - 34.

[6] Heidemann J S, Popek G J. File System Development with Stackable Layers[J]. ACM Transactions on Computer Systems, 1994, 12(1): 58 - 59.

[7] Hendricks D. A Filesystem For Software Development[C]//In Proceedings of the USENIX Summer Conference, Anaheim, CA: USENIX Association, 1990: 333 - 340.

[8] Roselli D, Lorch J R, Anderson T E. A Comparison of File System Workloads[C]//In Proceeding of the Annual USENIX Technical Conference, San Diego, CA: USENIX Association, 2000: 41 - 54.

[9] Maziéres D. A Toolkit for User - Level File Systems[C]//In Proceedings of the Annual USENIX Technical Conference, Boston, MA: USENIX Association, 2001: 261 - 274.

[10] Burnett N C, Bent J, Arpaci - Dusseau A C, et al. Exploiting Gray - Box Knowledge of Buffer - Cache Contents[C]//In Proceedings of the Annual USENIX Technical Conference, Monterey, CA: USENIX Association, 2002: 29 - 44.

(编辑:徐楠楠)

Study on the Security mechanism of Space FS

LIU Jin - Gang[1,2], LI Ning[1,2]

(1. Join Faculty of Computer Scientific Research Capital Normal University, Beijing 100037; 2. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

Abstract: By analyzing the problem of security and reliability existing in the traditional file system , this paper firstly proposes a security mechanism based on linux OS which means that the OS will be installed into a relatively independent space and separated with other software. Meantime, the phisical storage medium is only - read to make the OS un - rewritten. Users can install the software and save documents pellucidly in the user's space which is isolated with system space in phisical space. For the users, the views or the operation of the document system is as same as the traditional OS and the rock - bottom system space and user's space can not be felt. The principle of design and its practical details of space FS system are described particularly. Finally, the test data prove that the system is safe and reliable without performance loss.

Key words: Linux; space file system; secure and reliable

(本卷终)