基于 Isabelle/HOL 的文件系统形式化设计与验证

王文斌[1]，钱振江[1,2*]，靳勇[2]，孙高飞[2]，邢晓双[2]，苏超[2]，孙天琦[2]

（1.苏州大学计算机科学与技术学院，江苏 苏州 215000；2.常熟理工学院计算机科学与工程学院，江苏 苏州 215500）

摘　要：对于构建可信操作系统而言，文件系统设计和实现的正确性至关重要，即使是已经得到广泛运用的文件系统仍然有漏洞被检测出来。采用形式化方法对文件系统的设计和实现的正确性进行严格的验证是公认的可行方法。当前文件系统的形式化验证工作大多基于宏内核操作系统，而忽视了微内核操作系统架构下文件系统的验证。为此，提出一种微内核架构下采用内联数据机制的文件系统的形式化设计和验证方法。以高阶逻辑（HOL）和自动机模型为基础，将文件系统中的工作对象和系统资源抽象为系统对象来构建文件系统的工作状态，形式化地描述文件系统的相关系统调用的功能语义，将系统调用提供服务的过程抽象为系统工作状态发生跃迁的过程，并给出文件系统功能正确性和安全属性的断言。以实现的安全可信微内核操作系统（VSOS）中的安全可信文件系统（VSFS）为例，在设计阶段构建 VSFS 的有限状态机模型，并在 Isabelle/HOL 中抽象描述 VSFS 的可移植操作系统接口（POSIX）系统调用，分析和归纳出 VSFS 文件系统正确性断言，使用定理证明的方式来验证 VSFS 的正确性。实验结果表明，该方法在 Isabella/HOL 中完成 VSFS 有限状态机模型细粒度的形式化验证，满足预期的安全需求规范。

关键词：形式化验证；文件系统；定理证明；有限状态机；微内核

中图分类号：TP391　　**文献标志码**：A　　**DOI**：10.19678/j.issn.1000-3428.0067091

Formal Design and Verification of File System Based on Isabelle/HOL

WANG Wenbin[1], QIAN Zhenjiang[1,2*], JIN Yong[2], SUN Gaofei[2], XING Xiaoshuang[2], SU Chao[2], SUN Tianqi[2]

(1. School of Computer Science and Technology, Soochow University, Suzhou 215000, Jiangsu, China;
2. School of Computer Science and Engineering, Changshu Institute of Technology, Suzhou 215500, Jiangsu, China)

【Abstract】 To construct a trusted operating system, the correctness of file system design and implementation is essential. Even file systems that are widely used can have bugs. Using formal methods to verify the correctness of file system design and implementation is feasible. Most of the current formal verification research conducted on file systems are based on macro-kernel operating systems, whereas the verification of file systems under a micro-kernel operating system architecture is lacking. Accordingly, in this study, a formal design and verification method for a file system using an inline data mechanism with a micro-kernel architecture is proposed. Based on the Higher-Order Logic(HOL) and automaton models, the working state of the file system is constructed by abstracting the working objects and system resources in the file system into system objects, and the functional semantics of the related system calls of the file system are formally described. The process of invoking and providing the service is abstracted as the process of transitioning the system working state, and asserting the correctness of the file system function and security attributes is given. Using the implemented microkernel Verified Secure Operating System (VSOS) file system called Verified Secure File System(VSFS) as an example, the finite state machine model of VSFS is built in the design stage, Portable Operating System Interface of UNIX(POSIX) system of the VSFS is abstractly described in the Isabelle/HOL call, correctness assertion of the VSFS file system is analyzed and summarized, and a theorem proof is used to verify the correctness of VSFS. The experimental results depict that the proposed method can complete the fine-grained formal verification of the VSFS finite-state machine model in Isabella/HOL and meet the expected security requirements.

【Key words】 formal verification; file system; theorem proof; finite-state machine; microkernel

0　引言

文件系统作为操作系统中负责数据持久化存储的功能模块，即使是一个微小[1-3]的错误，其造成的损失也将是巨大的，因为这可能导致关键的数据永久丢失。实现一个安全可信的文件系统一直是操作系统开发人员的目标，近年来众多有关操作系统正确性方面的探索工作[4-6]已经显示出构建一个安全

收稿日期：2023-03-04　　**修回日期**：2023-06-26

基金项目：江苏省自然科学基金面上项目（BK20191475）；常熟市社会发展项目（CS202204）。

通信作者 E-mail：* qianzj@cslg.edu.cn

可信的文件系统的可行性。

在操作系统的开发过程中,文件系统的安全性是指:文件系统在设计阶段的设计符合预期的安全需求;文件系统的设计与实现是一致的;文件系统在开发过程中不存在各种语法错误等。在软件开发过程中,常见的用于保障软件安全性的方法有软件测试和静态分析。软件测试对于软件的安全性验证是不完整的,静态分析无法解决文件系统中的语义错误。因此,它们只能有效地满足最后一项要求,而无法满足文件系统安全性的前两项要求。

基于严格数理逻辑的形式化方法一直是公认的用于保障操作系统安全性的方法。早期由于缺少像 Isabelle/HOL 和 Coq 这样的辅助定理证明工具帮助完成文件系统形式化验证的自动推理工作,致使对文件系统进行全面的形式化验证是不现实的。另外,当前有关文件系统的形式化验证大多基于以 Linux 为代表的宏内核架构。宏内核将所有的系统服务都集成到内核中,导致在对文件系统进行形式化验证时可信计算基(TCB)过大。自从微内核架构在开发 Mach 系统时被提出以来,其在安全方面的特性便得到研究人员的关注。微内核架构在内核空间只提供进程调度、虚拟内存等最核心的功能,将大多数的系统服务以模块化的形式运行在用户态,以最大程度地降低内核的代码量。此特性可有效地降低形式化验证过程的 TCB 规模。seL4[7] 是第 1 个尝试经过完整形式化验证的微内核操作系统原型,但遗憾的是它并没有对运行在用户态的文件系统展开详细的形式化验证。

本文基于高阶逻辑和自动机模型中的有限状态机(FSA)理论,运用内联数据机制的微内核架构文件系统,提出一种细粒度的形式化设计和验证方法。通过抽象文件系统所涉及的工作对象和资源来构建文件系统的状态,根据文件系统的功能需求与安全属性给出相关系统调用的公理性语义,并将系统调用的公理性语义转换成有限状态自动机中的状态转移函数,以状态转移函数所造成的文件系统状态变化的合法性分析归纳出文件系统调用的功能正确性断言,以此在 Isabelle/HOL 中完成对于文件系统的建模和形式化验证。

1 相关工作

早期的文件系统形式化验证需要进行大量的定理证明工作[8],因此研究人员选择忽略底层实现细节,在较高的抽象层次去实现对于文件系统的形式化建模。通过对实现文件系统的高级语言进行限制,在开发过程中只允许使用特定优化过后的高级语言子集,可以在文件系统的设计规约与高级语言之间实现自动化链接,降低验证的工作量。文献[9-11]介绍的 C 语言的受限制子集 Cogent、用于数据布局细化证明的 DARFENT 以及美国宇航局在太阳轨道卫星系统的文件系统形式化验证中所使用的 Scala 受限制子集都是基于这一思想。一方面,这样的限制使得这些语言在开发时存在着功能受限的问题,如 Cogent 无法支持递归;另一方面,这些语言仅对其所实现的文件系统的设计规范与实现的一致性进行保证,而忽视了开发过程中所使用的语言组件的可信性验证。虽然文献[12]介绍的 Cogent-C 是针对后一种问题而提出的 Cogent 改良版本,其完成了文件系统开发过程中所使用的部分库的验证,但是仍然无法解决前一问题。

文献[13]介绍了上海交通大学研究人员开发的并发关系逻辑辅助验证(CRL-H)框架,并设计和验证了第 1 个经过形式化验证的并发文件系统 AutomFS。这是针对多核处理器与经过验证的并发式文件系统,但它不支持持久化数据,并且不考虑崩溃安全性。文献[14]通过使用局部思想和偏序法构建的 CRL-T,将其用于验证 AutomFS 的终止性的验证框架,成功地证明了并发性文件系统在公平调用情况下接口的终止性。

来自美国麻省理工学院(MIT)的计算机科学与人工智能实验室(CSAIL)的研究人员主持了一系列关于文件系统形式化验证项目,如表 1 所示。该项目组主要关注文件系统的崩溃安全与并发安全。文献[15]介绍的 FSCQ 是 CSAIL 针对文件系统崩溃安全而展开的文件系统形式化验证项目,它是使用 Coq 进行机器检查证明的文件系统。

表 1 文件系统形式化验证项目

Table 1 File system formal validation project

年份	项目名称	特征
2015	FSCQ	第 1 个被验证崩溃安全的文件系统
2016	Yggdrasil	完全自动的 SMT 推理工具
2017	DFSCQ	优化 FSCQ 的性能
2017	CFSCQ	使用乐观系统调用为 FSCQ 引入并发性
2018	SFSCQ	针对文件系统的并发机密性
2019	Argosy	支持模块化证明的分层恢复实现
2019	Perennial	具有恢复租约、恢复帮助和版本化内存三大特性
2020	IFSCQ	使用加密原语保护系统免受篡改
2021	GoJournal	第 1 个经过验证的使用 Go 语言实现的崩溃安全的文件系统
2022	DaisyNFS	使用 Go 语言实现的网络文件系统

为克服 FSCQ 性能较差的缺点，文献[16]介绍了后续的项目 DFSCQ，在以不破坏 FSCQ 崩溃安全为前提的情况下，通过为功能体 fsync 和 fdatafnc 提供精确的规约，为 FSCQ 引入这两种提高文件系统 I/O 速率的机制，进而开发出性能有所提高的文件系统 DSFCQ。文献[17]将加密文件系统与 FSCQ 文件系统相结合，设计了经过形式化验证的、使用加密原语保护磁盘文件安全的文件系统(IFSCQ)。文献[18]设计了名为 optimistic systems calls 的机制，以较小的工作量在 FSCQ 上引入并发机制，进而构建了并发性文件系统(CFSCQ)。针对并发文件系统机密性，文献[19]提出 DIESKSEC 方法，并开发了 SFSCQ 文件系统。文献[20]介绍了 Perennial 的形式化验证框架，对文件系统的并发性崩溃安全展开进一步的验证。文献[21-23]引入更多技术规范构建 Perennial 2.0 框架，并用其验证使用 Go 语言实现的网络文件系统服务器 DaisyNFS，以及它的日志服务系统 GoJournal。

CSAIL 的研究人员针对文件系统的形式化验证进行了诸多方向的探索和努力，他们的工作都是基于宏内核操作系统而展开的，在一定程度上无法隔绝操作系统其他内核模块对于文件系统的影响，因此在形式化验证过程中存在 TCB 过大的问题。

本文为一款用于网络通信、安全可信微内核操作系统(VSOS)[24]中的安全可信文件系统(VSFS)引入形式化的方法，探索在微内核架构下引入内联数据机制的可行性与实现的正确性。使用该方法在设计阶段构建 VSFS 的有限状态机模型，抽象描述有关 VSFS 的系统状态和可移植操作系统接口(POSIX)系统调用的功能语义，构建 VSOS 的形式化模型，在 Isabelle/HOL 中完成该模型的细粒度验证。

2 VSFS 文件系统

本文从 VSOS 架构、VSFS 总体布局、内联数据机制等方面来介绍 VSFS。对于 VSOS 架构特性和内核模块汇编级形式化验证工作，文献[24]中已有详细描述。本节重点介绍 VSFS 文件系统总体布局和适用于小文件场景的内联数据机制。

2.1 VSOS 架构

VSOS 是用于网络通信、安全可信的微内核架构操作系统。它在设计之初便致力于操作系统安全性，因此对内核模块进行了精心设计，最小化内核的代码量，其整体架构如图 1 所示。

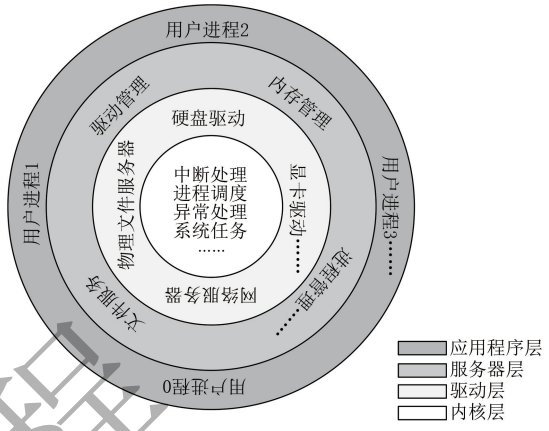

图 1 VSOS 整体架构

Fig.1 VSOS overall architecture

VSOS 在逻辑上被划分成 4 层：只提供中断处理、进程调度等少量核心服务的内核层；提供各硬件驱动服务的驱动层；VSFS 文件系统所在的服务器层，这一层向上为应用程序提供标准的系统调用，向下获取驱动服务和内核服务；最上层为应用程序层。在 VSOS 中内核层和服务器层经过形式化的验证[25]，确保在系统运行过程中不发生变化，因此被视为是安全可信的。高度个性化和具有极大自由度的驱动层和应用程序层，由于在运行时是动态加载的模块，将被视为是不可信的。

服务器层和驱动层都将作为独立的进程运行，它们之间只能通过进程间通信(IPC)来彼此间调用服务。如 VSOS 中提供的基础原语 SEND 和 RECEIVE。VSOS 通过提供一个轻量化、固定长度的消息结构来简化消息的处理。VSFS 运行在用户模式，被划分到服务器层。这将使得 VSFS 无法直接去获取内核中的特权服务，比如 I/O 操作。基于“最小权限原则”的 VSOS 在内核层设计了系统任务进程来专门处理 VSFS 通过 IPC 发出的内核调用，避免 VSFS 直接获取特权服务。

2.2 VSFS 设计布局

VSFS 在磁盘上设计布局如图 2 所示。VSFS 将 block 0 分配给 Boot 程序，负责程序的启动和运行。随后是磁盘上的元数据结构区域，即超级块、超级块备份、索引节点位图、块位图、索引节点树以及一个日志区域。在图 2 中它们的逻辑位置连续，但其在磁盘上的物理块位置不连续。超级块与超级块的备份并不会连续存储，因为连续存储会极大地增加两者同时损坏的概率。VSFS 将超级块的备份与日志区域都部署于磁盘存储位置的底部。日志、索引节点位图、块位图以及索引节点表的具体位置会记录在超级块中。

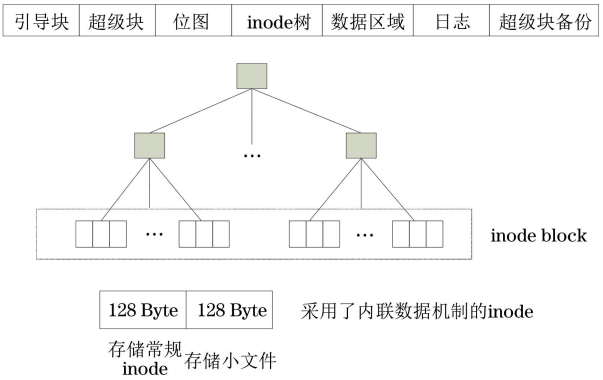

图 2 VSFS 文件系统的总体布局

Fig.2 General layout of VSFS file system

VSFS 会为每一个索引节点分配 256 Byte 的空间，但索引节点数据结构只使用 128 Byte，冗余的空间是为支持内联数据机制。

2.3 内联数据机制

基于提高处理体积较小文件效率、减少内核源码在运行时所占内存空间的目的，本文为 VSFS 引入了内联数据机制。VSFS 为每个索引节点分配 256 Byte 空间，但索引节点数据结构所使用的空间大小为 128 Byte。实际上索引节点数据结构使用的 128 Byte 空间中仍有部分未被使用，而是被当作扩展字段，为方便未来 VSFS 引进新的机制留有空间。内联数据机制会将不大于 128 Byte 的文件(后文将不大于 128 Byte 的文件称为小文件)存放在未被索引节点数据结构使用的 128 Byte 的索引节点空间中。

当启用内联数据机制时，系统会将较小的文件保存在索引节点的冗余空间中。在文件系统将文件从数据缓冲池中刷新回磁盘时，小文件若不采用内联数据机制，则磁盘需要寻找空闲数据块来存放该文件的数据。由于在 VSFS 中数据块缺省的大小为 4 KB，在不使用内联数据机制、单独存放小文件数据时，磁盘将会产生内部碎片。因此，采用内联数据机制会极大地减少磁盘内部碎片。

当文件系统读取小文件数据时，只需要遍历目录树找到该文件的索引节点，即可查询到该文件的数据，而无须通过索引节点再次查询数据所在位置。当数据缓冲未命中时，将节省时间开销。由于 VSFS 只需读取一次磁盘便可将索引节点数据和文件数据读入内核空间的索引节点缓存。若不采用内联数据机制，则 VSFS 第 1 次读磁盘会取出该文件的索引节点，再通过查询索引节点中记录的数据块的存放位置从磁盘上读取该文件数据，如图 3 所示。由于磁盘的读取速度和内存的读取速度存在数量级的差异，因此在这种情况下节省了时间开销。

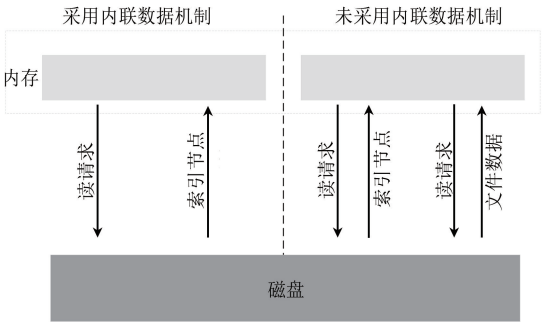

图 3 内联数据机制读取的对比

Fig.3 Comparison of inline data mechanism read

磁盘的硬件机制保证了磁盘上不大于一个扇区大小(即 512 Byte)的读写是原子性的，对于将索引节点数据结构和文件数据存放在一起的小文件来说，这使得在不使用任何额外软件机制辅助的前提下，即可保证对于小文件读写的原子性。

3 VSFS 文件系统的有限状态机模型

本节介绍基于高阶逻辑和自动机模型为 VSFS 构建形式化模型的过程，通过抽象 VSFS 文件系统相关的工作对象和资源来构建文件系统的状态。在此基础上，根据 VSFS 的功能需求与安全属性给出为上层应用程序提供的系统调用的公理性语义，并将系统调用的公理性语义转换成有限状态自动机中的状态转移函数，最后给出其在 Isabelle/HOL 中相应的定义。

本文使用的自动机模型为确定性的有限状态自动机。它的一般性定义为一个五元组，如式(1)所示：

$$M=(S,\Phi,\xi,S_1,S_E) \tag{1}$$

其中：S、S_1、S_E 分别表示系统的状态集合、系统的初始状态、系统的结束状态；Φ 是自动机可识别的字母表；ξ 是状态转移函数。

3.1 状态

VSFS 提供服务的过程可以看作 VSFS 中状态的转变过程。在理论上，VSFS 的状态集合 S_{vsfs} 是由 VSFS 中工作对象 $X_{i,j}$ 所形成的笛卡儿积空间，如式(2)所示：

$$S_{vsfs}=\prod_{i,j} \tag{2}$$

由于 VSFS 工作对象的取值有限，如索引节点数量，因此真实状态集合 S_1 是 S 的一个子集，即 $S_1\in S$。VSFS 的系统状态在 Isabelle/HOL 中的定义如算法 1 所示，其中，state 是一个 record 数据类型。

算法 1 VSFS 系统状态在 Isabelle/HOL 中抽象

```
record state =
superBlocks :: SUPER_BLOCK_INFO
filps :: "FILE list"
```

dentryCache :: "D_cache"
inodeCache :: "I_cache"
bufferCache :: "B_cache"
fprocs :: "FPROC list"
disk :: "disk"
messages :: "MESSAGE"

在算法 1 中，superBlocks 表示内存中超级块的抽象，它存放有 VSFS 文件系统的有关信息，filps 表示进程与被打开文件之间信息的抽象，文件描述符 fd 指向 filp 中被打开的文件，dentryCache 表示目录项在内存中的缓存抽象，由 3 个列表构成，分别为空闲目录项对象的缓存与 VSFS 在初始化时将预分配部分的页作为目录项的缓存，正在被使用的目录项对象，刚刚释放的目录项对象，类似于 Linux 中的最近最少使用(LRU)链表，inodeCache 表示索引节点在内存中的抽象，其结构类似于 dentryCache，bufferCache 表示存放 VSFS 从磁盘读取数据的抽象，在实现过程通过 buffer_head 来操作，buffer_head 记录各缓冲块的详细信息，fprocs 表示进程与文件系统之间相关信息的抽象，messages 表示进程之间 IPC 发送消息的抽象，disk 表示 VSFS 磁盘的抽象，它是 Isabelle/HOL 对于 VSFS 中磁盘总体布局中各个数据结构的抽象，其在 Isabelle/HOL 中的定义如算法 2 所示。inodeTable 记录磁盘上所有索引节点的集合。需要注意的是，为支持内联数据机制，索引节点由元数据和它所记录的内联数据两部分所构成。内联数据机制必须满足 2.3 节的要求才会被使用。

算法 2 磁盘在 Isabelle/HOL 中的抽象

record disk =
superBlocks :: D_SUPER_BLOCK
superBlockbk :: D_SUPER_BLOCK
Imap :: "int list"
Bmap :: "int list"
inodeTable :: "D_INODE list"
files :: "(NAME,INODE_INFO)file"

在算法 2 中，superBlocks 表示磁盘的超级块信息，superBlockbk 表示超级块在磁盘的备份，Imap 表示索引节点空闲位图，Bmap 表示磁盘块空闲位图，inodeTable 表示磁盘的索引节点表，files 表示磁盘文件是由目录树组织的，通过使用 file 类型结构来抽象。这是一个为特里树(trie)类型的抽象。叶子节点为普通的文件，树节点的子节点表示的是一个目录文件所包含的文件。每个树上的节点有文件名 NAME 和文件相关信息及内容 INOED_INFO 两个域。

至此，给出 VSFS 的状态集合 S_{vsfs} 定义，如式(3)所示：

$$S_{vsfs}=S_0 \cup S_1 \cup \cdots \cup S_{n-1} \cup S_n \tag{3}$$

其中：S_0 表示 VSFS 初始化之前的合法状态集合，即式(4)，vsfs_disk()表示磁盘检测函数；S_1 表示经过初始化后能够执行各种系统调用的状态的集合，即式(5)；S_n 表示执行完成各个系统调用后，可以接收下次消息请求的状态集合，即式(6)；S_{read}、S_{write} 分别表示 VSFS 准备提供读写服务时的状态集合；$S_2 \sim S_{n-1}$ 分别表示 VSFS 在执行各个请求之前的状态集合。以 read 请求为例，设执行读取服务的状态集合为 S_{read}，即式(7)。

$$\forall s \in S_0 . \text{vsfs\_disk}(s) \tag{4}$$

$$\forall s \in S_1 . \exists s' \in S_0 . \text{disk } s = \text{disk } s' \tag{5}$$

$$\forall s' \in S_n . ((\exists s' \in S_{read} . s = \text{vsfs\_read } s') \cup (\exists s' \in S_{write} . s = \text{vsfs\_write } s') \cup \cdots) \tag{6}$$

$$\forall s \in S_{read} . \text{m\_type}(\text{message } s) = \text{READ} \tag{7}$$

初始状态 $S_1 \in S_0$，即表示 VSFS 的初始状态应当通过合法的磁盘检查；终止状态 $S_E = S_1 \cup S_n$，即 VSFS 的终止状态为执行完系统调用后各种状态的并集。

3.2 Φ_{vsfs} 字母表

Φ_{vsfs} 是 VSFS 抽象模型可以识别的字母表，它由 VSFS 的系统调用所构成，其定义如式(8)所示：

$$\Phi_{vsfs}=\{i, \text{do\_}r, r, \cdots\} \tag{8}$$

其中：i 表示初始过程；do_r 表示 VSFS 提供给上层应用程序所使用的系统调用接口；r 表示 VSFS 提供实际读取服务的事件。其他提供给上层服务的系统调用接口与实际提供服务的事件也以 do_x、x 的形式来表示，x 为提供给应用程序层的系统调用。

3.3 状态转移函数 ξ_{vsfs}

根据有限状态自动机的定义，VSFS 的状态转移函数 ξ_{vsfs} 的定义如式(9)所示：

$$\xi_{vsfs} : S_{vsfs} * \Phi_{vsfs} \rightarrow S_{vsfs} \tag{9}$$

ξ_{vsfs} 表示当 VSFS 系统状态处于 s ($s \in S_{vsfs}$) 时，发生事件 e ($e \in \Phi_{vsfs}$)，状态转移函数 Δ ($\Delta \in \xi_{vsfs}$)将会通过修改抽象模型中的工作对象，使得 VSFS 系统状态转变为 s' ($s' \in S_{vsfs}$)。

状态转移函数由 VSFS 文件系统为用户应用层提供的系统调用抽象而来。基于 Hoare 逻辑三元组来构建这些函数的公理语义，如式(10)所示：

$$\{P\}F\{Q\} \tag{10}$$

式(10)表示程序 F 在入口处与出口处各变量满

足的条件,即为状态跃迁的前后状态,P 和 Q 分别为前置条件与后置条件。当前置状态合法时,功能正确的系统调用在运行结束后,后置状态也应当合法。以初始化函数 vsfs_init()为例进行讲解。vsfs_init()所对应的事件为 i,设在 VSFS 初始化前的状态为 s_0,初始化后的状态为 s'。vsfs_init(s_0,i) = s'表示系统启动后可以正确初始化并达到统一状态。vsfs_init()公理语义如式(11)所示:

$$\{tt\}vsfs\_init\{R(s,s')\} \tag{11}$$

其中:tt 为永真谓词,表示 VSFS 处于任何状态。式(11)表示无论执行初始化服务之前系统处于何种状态,经初始化后都将满足条件 R。R(s,s')的定义如算法 3 所示。

算法 3 后置条件 R(s , s')

(s'. superBlocks= s. disk. superBlocks) ∧

(s'. dentryCache = (freeDentrylst = (FreeDentry ♯ … ♯ FreeDentry) ∧ (inuseDentrylst = [RootDentry]) ∧ (unuseDentrylst = [])) ∧

(s'. inodeCache = …) ∧ (s'. bufferCache = …) ∧ (s'. filps = Freefile ♯ … ♯ Freefilp) ∧

(s'. fprocs = []) ∧ (s'. disks = s. disks)

dentryCache、inodeCache、bufferCache 由 3 个对应的缓存列表构成,初始化时将根节点的信息更新至缓存。inodeCache 与 bufferCache 的语义类似于 dentryCache 的语义。根据 vsfs_init()的公理语义,在 Isabelle/HOL 中给出它的形式化抽象,如算法 4 所示。

算法 4 vsfs_init()在 Isabelle/HOL 中的抽象

```
fun vsfs_init::"state ⇒ state"
where
"vsfs_init s = s(|
superBlocks:= sb_init (superBlock(disk s)),
filps:= init_filp_s (filp_s s),
dentryCache:= d_cache_init (dentry_cache s),
inodeCache:= i_cache_init (inode_cache s),
bufferCache:= b_cache_init (buffer_cache s),
fprocs:= init_fproc_list [] FPROC_NUM,
disk:= disk s|)"
```

vsfs_init()的类型为"state⇒state"。在抽象过程中使用辅助函数来更加简洁高效地表示。以 sb_init()为例,它所需的参数为磁盘超级块信息(superBlock(disk s)),返回的对象被用于更新工作对象 superBlocks,其他的辅助函数与此类似。这使得在后续验证过程中,必须验证辅助函数的正确性。

以文件系统最重要的提供读写服务的 write()、read()系统调用为例,两者实现函数分别为 vsfs_write()、vsfs_read(),其在 Isabelle/HOL 中的抽象分别为算法 5、算法 6。

算法 5 vsfs_write()在 Isabelle/HOL 中的抽象

```
fun vsfs_write::"state ⇒ state"
where
"vsfs_write s =
(if valid_fd s (m_fd(messages s))
then (if valid_write_len s
then (if write_free_block s
then (if write_free_buffer s
then (if inline_data s
then inline_write s
else plain_write s)
else s) else s) else s) else s)"
```

算法 6 vsfs_read()在 Isabelle/HOL 中的抽象

```
fun vsfs_read::"state ⇒ state"
where
"vsfs_read s =
(if valid_fd s (m_fd(message_s s))
then (if m_len(message_s s) > 0
then (if find_free_buffer s
then (if valid_pos s
then (if if_inline_data s
then inline_read s
else plain_read s )
else s) else s) else s) else s)"
```

读写服务的具体实现分为两种:在读写文件时,会通过索引节点中的 inline_type 字段识别是否启用内联数据机制,若启用内联数据机制,则在读取数据时会先检索索引节点的后 128 Byte 空间;在写入文件时,会先判断新数据写入是否导致文件大小超出内联数据机制的容量限制,若未超出则将数据写至索引节点的后 128 Byte 空间,若超出则为其分配新数据块,将数据复制到该数据块,并修改相关标识位。

VSFS 文件系统为上层应用程序提供的系统调用都有与之对应的状态转换函数,根据它的功能规范与安全需求建立与之对应的公理语义,并在 Isabelle/HOL 中实现状态转移函数的抽象。

至此,以一个五元组来表示 VSFS 的有限状态机模型 M_{vsfs},如式(12)所示:

$$M_{vsfs} = (S_{vsfs}, \Phi_{vsfs}, \xi_{vsfs}, S_I, S_E) \tag{12}$$

4 正确性证明

VSFS 为上层用户程序提供的服务是其提供的系统调用集合,因此为验证 VSFS 的安全性与可信性,不仅要验证在抽象系统调用时所使用的辅助函

数功能正确性，还需归纳分析出 VSFS 有限状态机模型中抽象化的状态转移函数正确性断言。通过使用证明策略，在交互式定理辅助证明器 Isabelle/HOL 中完成对于辅助函数功能正确性和状态转移函数正确性的断言验证。

4.1 辅助函数的功能正确性

为更加细粒度地验证文件系统的实现正确性，以及提高验证工作的模块化与简洁性，本文使用了大量的辅助函数。针对 VSFS 的正确性，这些辅助函数的功能正确性同样决定着整个模型的正确性。因此，对于这些辅助函数的功能正确性的验证是必要的。

以磁盘上的文件抽象 file 为例，定义如式(13)所示：

$$f_{\text{file}}=(T_{\text{f}},\xi_{\text{f}},\rho_{\text{f}}) \tag{13}$$

T_{f} 为文件目录树的抽象结构，它在 Isabelle/HOL 中的定义如算法 7 所示。

算法 7 T_{f} 在 Isabelle/HOL 中的定义

datatype ('a,'v)File = FILE"'v option""('a * ('a,'v)File) list"

ξ_{f} 为作用在 T_{f} 上的操作函数，即为抽象系统调用过程中使用的辅助函数，其定义如式(14)所示：

$$\xi_{\text{f}}=(\text{getFileName},\text{getDir},\text{isFile},\text{isDir},\text{updateDir}) \tag{14}$$

这些函数的功能分别为返回文件名、返回目录结构、查询指定文件是否存在并返回、指定目录是否存在并返回、更新目录结构中的内容。

ρ_{f} 为 ξ_{f} 的安全属性，其定义如式(15)所示：

$$\rho_{\text{f}}=(\text{P1},\text{P2},\text{P3},\text{P4},\text{P5}) \tag{15}$$

其中：P1、P2、P3、P4、P5 分别为 ρ_{f} 中对应函数的功能正确性的属性。具体定义见表 2。

表 2 ρ_{f} 中各属性的具体定义

Table 2 Specific definitions of each attributes in ρ_{f}

属性	描述
P1	lemma A1: "getFileName(FILE ov al) = ov"
P2	lemma A2: "getDir(FILE ov al) = al"
P3	lemma A3: "∀ lst as bs. isFile (as≠lst) bs = (let (*a*,*b*) = as in if *a*=bs then Some *b* else isFile lst bs)"
P4	lemma A4: "isDir (FILE None []) as = None"
P5	lemma A5: "∀ *t v* bs. isDir (updateDir *t* as *v*) bs= (if as=bs then Some *v* else isDir *t* bs)"

由于属性 P1、P2、P3 的内容并不复杂，根据它们各自所对应的函数的定义，在 Isabelle/HOL 中使用自动证明策略“auto”完成证明。属性 P4 与属性 P5 的证明策略较为类似，以更为复杂的属性 P5 为例进行讲解。属性 P5 所对应的函数 updateDir 通过递归方式查找文件目录树来更新文件内容，因此通过对变元 as 使用证明策略“induct_tac”进行启发式归纳，获得证明子目标 g1、g2，如表 3 所示。

表 3 属性 P5 证明过程中各子目标

Table 3 Each subgoal in the proof process of attribute P5

名称	子目标具体内容
g1	∀ *t v* bs. isDir (updateDir *t* [] *v*) bs=(if []=bs then Some *v* else isDir *t* bs)
g2	∧a list. ∀ *t v* bs. isDir (updateDir *t* list *v*) bs = (if list = bs then Some v else isDir t bs) ⇒ ∀ *t v* bs. isDir (updateDir *t* (*a* # list) *v*) bs =(if *a* # list = bs then Some *v* else isDir *t* bs)
g3	∧ *t v bs*. [] ≠bs⇒ isDir (FILE (Some *v*) (getDir *t*)) bs = isDir *t* bs
g4	∧a list *t v* bs. ∀ *t v* bs. isDir (updateDir *t* list *v*) bs=(if list=bs then Some *v* else isDir *t* bs) ⇒ isFile (getDir *t*) *a* = None ⇒*a* # list≠bs⇒ isDir (FILE (getFileName *t*) ((*a*, updateDir (FILE None []) list *v*) # getDir *t*)) bs = isDir *t* bs
g5	∧a list *t x*2 *v* bs. ∀ *t v* bs. isDir (updateDir *t* list *v*)bs=(if list =bs then Some *v* else isDir *t* bs) ⇒ isFile (getDir *t*) *a* = Some *x*2 ⇒*a* # list≠bs⇒ isDir (FILE (getFileName *t*) ((*a*, updateDir *x*2 list *v*) #getDir *t*)) bs=isDir *t* bs

在 Isabelle/HOL 的帮助下使用证明策略“auto”可以将证明工作转化为证明子目标 g3、g4、g5。最后使用证明策略“case_tac”对变元 bs 进行分类实例化，即可在证明策略“auto”的帮助下完成对于属性 P5 的证明，在 Isabelle/HOL 中“No subgoals”表示完成证明。具体证明过程与结果如图 4 所示。

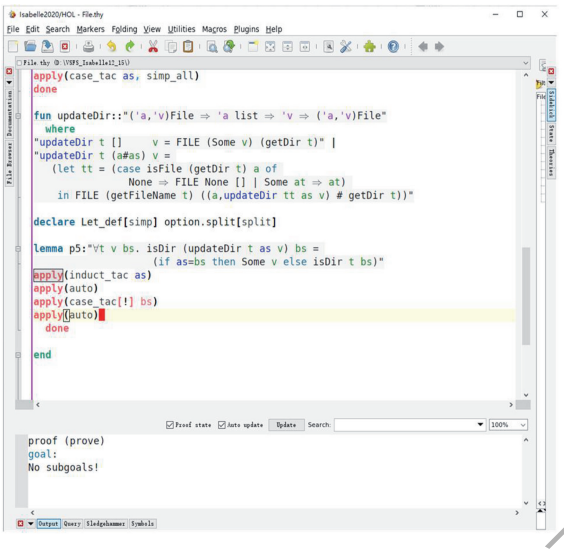

图 4 属性 P5 在 Isabelle/HOL 中的证明过程

Fig.4 The proof process of attribute P5 in Isabelle/HOL

同理,其他系统调用的属性也通过在 Isabelle/HOL 中使用类似的证明策略来完成正确性证明。

4.2 状态转移函数的断言

在分析归纳出状态转移函数断言之前,需要先给出 VSFS 的不变式 $V(s)$。不变式 $V(s)$ 的具体含义是指磁盘上的数据需要与内存上缓存的对应数据保持一致性。假设 VSFS 在状态 s 提供相应的系统调用后,系统的状态跃迁为 s'',该一致性条件仍然成立,即 $V(s'')$ 仍然成立。不变式是由磁盘格式的正确性属性以及内存上 inode、dentry、buffer 3 个对象的缓存链表的正确性属性来决定的。它们在 Isabelle/HOL 中的抽象分别对应 vsfs_inode()、vsf_dentry()、vsfs_buffer()。

以最为复杂的磁盘格式的正确性属性为例,其表示需要检查磁盘上整体数据的正确性,如超级块中魔数是否为指定值、第 1 块可用的数据块号是否合法、在初始化时超级块中数据与超级块备份数据是否一致等。此外,磁盘格式的正确性属性对磁盘的文件组织关系也有所限制,如不同的文件所具有的数据块号相异、处于不同目录下文件的索引节点号相异、文件所持有的索引节点号与数据块号分别在索引节点位图和块位图中对应数据位都已被置位等。

有了不变式的定义,接下来给出 VSFS 中各系统调用的正确性断言。如表 4 所示。表中给出了提供初始化服务的 vsfs_init()断言 A1、提供读取服务的 vsfs_read()的断言 A2,以及真正实现读取服务的 2 个不同函数 Plain_read()和 Inline_read()的正确性断言 A3 和 A4。

如前所述,这些断言在 Isabelle/HOL 的帮助下,通过使用证明策略完成正确性验证。同理,其他系统调用的正确性断言也通过类似的方法进行形式化证明。

表 4 VSFS 文件系统中部分正确性断言的定义

Table 4 Definition of partial correctness assertion in VSFS file system

断言	描述
A1	Theorem vsfs_Init_Correctness: $"\forall s \in S_0.$ (vsfs_disk(disk s)) $\rightarrow$ vsfs _disk(disk(vsfs_init s)) $\wedge$ vsfs _buffer(vsfs_init s) $\wedge$ vsfs _dentry(vsfs_init s) $\wedge$ vsfs _inode(vsfs_init s)"
A2	Theorem vsfs_Read_Correctness: $"\forall s \in S_1.$ (vsfs _disk(disk s)) $\wedge$ m_type (message_s s) = READ $\rightarrow$ vsfs _disk(disk (vsfs_read s)) $\wedge$ vsfs _buffer(vsfs_read s) $\wedge$ vsfs _dentry(vsfs_read s) $\wedge$ vsfs _inode(vsfs_read s)"
A3	Theorem Plain_Read_Correctness: $"\forall s \in S_1.$ (vsfs _disk(disk s)) $\wedge$ vsfs_buffer $s \wedge$ vsfs _dentry $s \wedge$ vsfs _inode $s \wedge$ inline_type=0 $\rightarrow$ vsfs _disk(disk (plain_read s)) $\wedge$ vsfs _buffer(disk (plain_read s)) $\wedge$ vsfs _dentry(disk (plain_read s)) $\wedge$ vsfs _inode(disk (plain_read s))"
A4	Theorem Inline_Read_Correctness: $"\forall s \in S_1.$ (vsfs _disk(disk s)) $\wedge$ vsfs_buffer $s \wedge$ vsfs _dentry $s \wedge$ vsfs _inode $s \wedge$ inline_type=1 $\rightarrow$ vsfs _disk(disk(inline_read s)) $\wedge$ vsfs _buffer(disk(inline_read s)) $\wedge$ vsfs _dentry(disk(inline_read s)) $\wedge$ vsfs _inode(disk(inline_read s))"
…	…

5 结束语

本文基于高阶逻辑和有限状态机理论,提出一种细粒度的形式化方法对微内核架构操作系统的文件系统模块进行设计和验证,并为安全可信的微内核操作系统 VSOS 设计和验证了带有内联数据机制的文件系统 VSFS。在 Isabelle/HOL 中,通过定理证明的方法完成了对 VSFS 功能正确性验证。下一步将对微内核操作系统的文件系统的并发性进行研究,以提高文件系统在提供并发性数据服务时的可靠性和安全性。

参考文献

[1] ETHI U, PAN H, LU S, et al. Cancellation in systems: an empirical study of task cancellation patterns and failures[C]// Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA: USENIX Association, 2022: 127-141.

[2] OU X, YUAN J, SHILANE P, et al. The dilemma between deduplication and locality: can both be achieved? [C]// Proceedings of the 18th USENIX Conference on File and Storage Technologies. Santa Clara, USA: USENIX Association, 2021: 171-185.

[3] PITCHUMANI R, KEE Y S. Hybrid data reliability for emerging key-value storage devices[C]//Proceedings of the 18th USENIX Conference on File and Storage Technologies. New York, USA: ACM Press, 2020: 309-322.

[4] SONG Y, CHO M, KIM D, et al. CompCertM: CompCert with C-assembly linking and lightweight modular verification [C]// Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, USA: ACM Press, 2019: 1-23.

[5] 李青,朱晓冉,郭建. AUTOSAR OS存储保护机制的形式化验证框架[J]. 计算机工程, 2017, 43(1): 79-85.
LI Q, ZHU X R, GUO J. Formal verification framework for AUTOSAR OS storage protection mechanism[J]. Computer Engineering, 2017, 43(1): 79-85. (in Chinese)

[6] 石剑君,计卫星,石峰. 操作系统内核并发错误检测研究进展[J]. 软件学报, 2021, 32(7): 2016-2038.
SHI J J, JI W X, SHI F. Recent progress of concurrency bug detection in operating system kernels [J]. Journal of Software, 2021, 32(7): 2016-2038. (in Chinese)

[7] KLEIN G, ELPHINSTONE K, HEISER G, et al. seL4: formal verification of an OS kernel[C]//Proceedings of the 22nd ACM SIGOPS Symposium on Operating systems Principles. New York, USA: ACM Press, 2009: 207-220.

[8] 杨龙婴,郭宇. 针对NAND闪存硬件的形式化建模[J]. 计算机工程, 2015, 41(11): 94-99.
YANG L Y, GUO Y. Formal modeling for NAND flash hardware[J]. Computer Engineering, 2015, 41(11): 94-99. (in Chinese)

[9] AMANI S, HIXON A, CHEN Z L, et al. Cogent: verifying high-assurance file system implementations[C]//Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2016: 175-188.

[10] CHEN Z L, LAFONT A, O'CONNOR L, et al. Dargent: a silver bullet for verified data layout refinement [J]. Proceedings of the ACM on Programming Languages, 2023, 7(1): 47.

[11] HAMZA J, FELIX S, KUNCAK V, et al. From verified scala to STIX file system embedded code using stainless[C]// Proceedings of the 14th International NASA Formal Methods Symposium. Pasadena, USA: [s. n.], 2022: 393-410.

[12] CHEUNG L, O'CONNOR L, RIZKALLAH C. Overcoming restraint: composing verification of foreign functions with cogent[C]//Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs. New York, USA: ACM Press, 2022: 13-26.

[13] ZOU M, DING H R, DU D, et al. Using concurrent relational logic with helpers for verifying the AtomFS file system[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. New York, USA: ACM Press, 2019: 259-274.

[14] 邹沫,谢昊彤,魏卓然,等. 基于锁耦合遍历算法的文件系统终止性验证[J]. 软件学报, 2022, 33(8): 2980-2994.
ZOU M, XIE H T, WEI Z R, et al. Verification of termination for file system based on lock coupling traversal[J]. Journal of Software, 2022, 33(8): 2980-2994. (in Chinese)

[15] CHEN H G, ZIEGLER D, CHAJED T, et al. Using Crash Hoare logic for certifying the FSCQ file system [C]// Proceedings of the 25th Symposium on Operating Systems Principles. New York, USA: ACM Press, 2015: 18-37.

[16] CHEN H G, CHAJED T, KONRADI A, et al. Verifying a high-performance crash-safe file system using a tree specification[C]//Proceedings of the 26th Symposium on Operating Systems Principles. New York, USA: ACM Press, 2017: 270-286.

[17] ONG D W, MAMOURAS K, CHEN A. The design and implementation of a verified file system with end-to-end data integrity[EB/OL]. [2023-02-01]. http://arxiv.org/abs/2012.07917v1.

[18] CHAJED T, CHLIPALA A, KAASHOEK M, et al. Extending a verified file system with concurrency [D]. Houston, USA: Rice University, 2017.

[19] LERI A M, CHAJED T, CHLIPALA A, et al. [C]// Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation. New York, USA: USENIX Association, 2018: 323-338.

[20] CHAJED T, TASSAROTTI J, KAASHOEK M F, et al. Verifying concurrent, crash-safe systems with Perennial[C]// Proceedings of the 27th ACM Symposium on Operating Systems Principles. New York, USA: ACM Press, 2019: 243-258.

[21] CHAJED T, TASSAROTTI J, THENG M, et al. Verifying the DaisyNFS concurrent and crash-safe file system with sequential reasoning[C]//Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA: USENIX Association, 2022: 447-463.

[22] HAJED T, TASSAROTTI J, THENG M, et al. GoJournal: a verified, concurrent, crash-safe journaling system[C]// Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA: USENIX Association, 2021: 423-439.

[23] HAJED T. Verifying a concurrent, crash-safe file system with sequential reasoning [D]. Cambridge, USA: Massachusetts Institute of Technology, 2022.

[24] QIAN Z J, ZHONG S, SUN G F, et al. A formal approach to design and security verification of operating systems for intelligent transportation systems based on object model[J]. IEEE Transactions on Intelligent Transportation Systems, 2023, 24(12): 15459-15467.

[25] QIAN Z J, XIA R, SUN G F, et al. A measurable refinement method of design and verification for micro-kernel operating systems in communication network [J]. Digital Communications and Networks, 2023, 9(5): 1070-1079.

编辑 索书志