2013 年 10 月 第 10 期
Vol. 34 No. 10 2013

微内核架构文件系统的形式化设计与验证方法研究

钱振江[1,2,3,4],唐洪英[2,3],李康杰[2,3],黄 皓[2,3],宋方敏[2,3]

[1](常熟理工学院 计算机科学与工程学院,江苏 苏州 215500)
[2](南京大学 软件新技术国家重点实验室,南京 210046)
[3](南京大学 计算机科学与技术系,南京 210046)
[4](伦敦大学 国王学院,英国 伦敦 WC2R 2LS)
E-mail: zhenjiang. qian@ gmail. com

摘 要: 文件系统作为数据存储和管理的功能模块,其正确性是操作系统安全性的重要方面. 采用形式化方法对微内核架构文件系统进行设计,使用操作系统对象语义模型(OSOSM)框架提出微内核架构文件系统的状态自动机模型,并依此描述系统调用的功能语义和系统状态转换,分析和归纳文件系统的功能正确性断言. 以实现的微内核安全操作系统(Verified Trusted Operating System,VTOS)为例,阐述在 Isabelle/HOL 定理证明器环境中构建状态自动机模型的方法,并对 VTOS 文件系统的形式化设计和功能正确性断言进行一致性验证,结果显示,VTOS 文件系统的设计和实现符合预期的正确性规格说明.

关 键 词: 文件系统; 微内核架构; 形式化设计; 形式化验证; 正确性断言; Isabelle/HOL
中图分类号: TP316 **文献标识码:** A **文 章 编 号:** 1000-1220(2013)10-2261-06

Research on Methodology of Formal Design and Verification for File System Based on Microkernel Architecture

QIAN Zhen-jiang[1,2,3,4], TANG Hong-ying[2,3], LI Kang-jie[2,3], HUANG Hao[2,3], SONG Fang-min[2,3]
[1](School of Computer Science and Engineering, Changshu Institute of Technology, Suzhou 215500, China)
[2](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210046, China)
[3](Department of Computer Science and Technology, Nanjing University, Nanjing 210046, China)
[4](King's College London, London WC2R 2LS, UK)

Abstract: The file system is the functional module for data storage and management, and its correctness is the key aspect of system security. In this paper, based on the operating system object semantics model (OSOSM), we propose a state automaton model for file system on the microkernel architecture. Based on state automaton model, we formally describe the functional semantics of system calls and the state transitions of file system, and analyze and define the system correctness assertions. We take the self-implemented microkernel security operating system (Verified Trusted Operating System, VTOS) as the example to illustrate the method to construct the state automaton model in the Isabelle/HOL theorem prover, and verify the consistency of formal design and correctness assertions of file system in VTOS. The result shows that the design and implementation of file system in VTOS correspond with the expected correctness specifications.

Key words: file system; microkernel architecture; formal design; formal verification; correctness assertion; Isabelle/HOL

1 引 言

文件系统(File System,FS)作为操作系统(Operating System, OS)中数据存储和管理的功能模块,其正确性是整个系统的安全性的重要基础. 如何保证 FS 的设计和实现的正确性一直是 OS 领域研究的热点. 采用严格的形式化方法对 FS 进行设计和实现是公认的可靠方法. 很多学者进行了相关研究: Morgan C 和 Sufrin B 从数学集合论(set theory)的角度对 UNIX 早期的文件系统(UNIX Filing System)进行了形式化的描述和验证[1],抽象的层次为系统调用(system call)级; Wenzel M 和 München TU 对 UNIX 文件系统的安全模型进行了形式化的建模[2],并在此基础上对 UNIX 的部分安全策略和机制进行了验证; Damchoom K 等人采用 Event-B[3] 和 Rodin[4] 平台对一款树状结构的文件系统 FS 进行了形式化的建模和验证[5],形式化工作主要关注该 FS 中改变树状结构的系统操作,如 create/copy/delete/move 等,并且采用"逐层精化"的方法对系统进行抽象和描述; Hesselink W H 等人采用 PVS[6] 对一款抽象文件系统进行了形式化验证[7],该抽象文

收稿日期: 2013-06-19 收修改稿日期: 2013-08-15 基金项目: 国家"八六三"高技术研究发展计划项目(2011AA01A202)资助; 国家自然科学基金创新研究群体基金项目(60721002)资助; 江苏省"六大人才高峰"高层次人才项目(2011-DZXX-035)资助; 江苏省高校自然科学基金项目(12KJB520001)资助. 作者简介: 钱振江,男,1982 年生,博士,讲师,研究方向为操作系统安全与形式化验证、嵌入式系统; 唐洪英,女,1988 年生,硕士,研究方向为操作系统安全、形式化验证; 李康杰,男,1985 年生,硕士,研究方向为操作系统安全、形式化验证; 黄 皓,男,1957 年生,博士,教授,博士生导师,研究方向为系统软件、信息安全; 宋方敏,男,1961 年生,博士,教授,博士生导师,研究方向为符号逻辑、量子计算机.

件系统采用偏函数(partial function) 的方式来描述和定义文件绝对路径,在具体的建模和验证过程中采用分层次的方法从 FS 的高层进行描述,重点验证系统操作符合预期的规格说明; Gallway A 等人采用 SPIN [8] 和 SMART [9] 工具以模型检测(Model-Checking) 的方法对 Linux 虚拟文件系统进行了形式化验证 [10] ,旨在对 Linux 虚拟文件系统的实际运行情况进行静态的分析,并对数据完整性等约束条件的可满足性进行验证.

本文认为,为了最大程度地保证 FS 的正确性,需要在 OS 的设计阶段就采用形式化方法构建 FS 模型,以此来指导系统的设计和实现.

本文采用形式化方法对微内核架构 FS 进行设计,提出微内核架构 FS 的状态自动机模型,并依此描述 FS 初始化、系统服务例程的功能语义和系统状态转化函数,分析和归纳 FS 的功能正确性断言. 同时,我们以实现的微内核安全操作系统(Verified Trusted Operating System,VTOS) 为例,在 Isabelle/HOL [11] 定理证明器环境中构建状态自动机模型,并对形式化设计和功能正确性断言进行一致性验证.

2 VTOS 文件系统

VTOS 是按照预期的功能和安全需求而实现的通用 OS. VTOS 采用微内核架构,只有微内核以特权级(0 级) 在核心态(Kernel mode) 运行,系统的其他功能服务模块在用户态(User mode) 运行,这样的一种架构保证了微内核作为最小可信基的安全性. VTOS 中各个功能模块之间通过消息(message) 的方式来进行通信.

在阐述对微内核架构 FS 的形式化设计和验证方法之前,本节首先说明 VTOS FS 模块的主要功能的设计思想.

在 VTOS 中,FS 功能模块作为处于用户态的服务进程,在系统加载时,从系统镜像加载到内存,并被加入到进程就绪队列,等待调度执行.

FS 模块的主要功能包括 FS 初始化(*fs_init*) 、文件打开(*fs_open*) 、文件关闭(*fs_close*) 、文件读取(*fs_read*) 、文件写入(*fs_write*) 服务例程等. 限于篇幅,本文主要从 *FS* 初始化(*fs_init*) 、文件读取(*fs_read*) 和文件写入(*fs_write*) 服务例程三个方面对 FS 的形式化设计和验证进行阐述.

在初始化(*fs_init*) 期间,FS 完成的工作主要包括两部分: 一是通过硬盘驱动,将磁盘超级块(super_block) 、FS 根索引节点(root inode) 读入内存,并为根索引节点建立目录项缓存(dentry) ; 二是为在系统初始化期间加载的进程创建 FS 相关信息,如数据结构 fproc,其中包括进程的当前工作目录、根目录,以及打开文件等信息.

FS 完成初始化之后,循环接收上层应用程序发来的文件操作请求(例如读取、写入等) . 考虑到系统的兼容性,VTOS FS 为应用程序提供的系统接口符合 POSIX 标准. FS 在接收到用户读取请求后,首先对请求的合法性进行检查,若请求合法,则 FS 先将用户请求的数据由磁盘读到 FS 缓存中,然后拷贝到用户空间,并返回读取的字节数; 否则,返回错误信息给用户,例如请求读的字节数小于 0. 对应地,FS 在处理写入请求时,首先对消息的合法性进行检查,若合法,则 FS 首先将用户需要写的数据拷贝到 FS 空间的缓存中,然后通过硬盘驱动程序将数据写到磁盘中; 否则,返回错误信息给请求服务的用户.

3 VTOS 文件系统的状态自动机模型和形式化描述

在上一节对 VTOS 文件系统框架描述的基础上,本节采用 OS 对象语义模型(OSOSM) [12] 来对 VTOS 文件系统进行形式化建模. OSOSM 是一种针对微内核 OS 的对象语义进行描述的开放式框架,将系统中的行为主客体抽象为对象,并从对象之间相互作用的角度表达系统的功能语义. 本文正是利用 OSOSM 的这一特性,将 VTOS FS 中的主客体进行抽象,建立 FS 的状态自动机模型,并依此描述 FS 初始化和服务例程的功能语义以及系统状态转化函数,分析和归纳 FS 的功能正确性断言. 同时在 Isabelle/HOL 定理证明器环境中构建上述的状态自动机模型,并对形式化设计和功能正确性断言进行一致性验证.

在对 FS 形式化建模之前,首先介绍 Isabelle/HOL 定理证明工具.

3.1 Isabelle/HOL 符号系统

Isabelle 是一个交互式的定理证明工具,Isabelle/HOL 是对 Higher-Order Logic 的支持. Isabelle 提供了这样的一套环境,将数学公式以一种逻辑语言的方式来表达,并提供了相应的工具以逻辑验算的方式对数学公式进行验证,可以用于验证软/硬件的设计正确性以及各种应用系统的安全属性等. 表 1 列出了本文形式化验证将用到 Isabelle/HOL 语法.

表 1 Isabelle/HOL 语法

Table 1 Isabelle/HOL syntax

语 法	说 明
int, nat, list, set	整数、自然数、列表和集合
type_synonym	定义数据类型别名
datatype	复合数据类型定义
record A =	
a1 :: 'a	记录类型定义
a2 :: 'b	
fun	全函数定义
primrec	原始递归函数定义
'a = > 'b = > 'c	函数类型
lemma, theorem	引理、定理定义
proof	证明过程
apply	证明规则应用

下面的章节描述 VTOS 文件系统的状态自动机模型的构建方法.

3.2 VTOS 文件系统的对象抽象及其状态空间

FS 中的对象由系统存储的数据表示,如文件系统磁盘超级块(super_block) 、文件索引节点(inode) 的缓存、目录项(dentry) 缓存、磁盘数据块在 FS 空间的缓存(buffer) 、打开文件指针、当前运行进程的 FS 部分描述结构、消息体、磁盘信息等. 我们将这些数据对象的结合作为系统的状态 state,为此具体定义如下:

```
record state =
    fs_super_blocks :: "fs_super_block list"
    fs_buffers :: "fs_free_buffer_list
        | fs_buffer_inuse_list
        | fs_buffer_unused_list"
    fs_dentries :: "fs_free_dentry_list
        | fs_dentry_inuse_list
        | fs_dentry_unused_list"
    fs_inodes :: "fs_free_inode_list
        | fs_inode_inuse_list
        | fs_inode_unused_list"
    fs_msgs :: "fs_msg list"
    fs_fprocs :: "fs_fproc list"
    fs_filps :: "fs_filp list"
    fs_disks :: "disk_data"
…
```

FS 状态是一个记录类型,其成员包括如下内容:

1) FS 磁盘超级块信息 *fs_super_blocks*,类型是系统中所有已安装文件系统的磁盘超级块(*fs_super_block*)组成的列表;

2) 磁盘数据块在 FS 空间的缓存信息 *fs_buffers*,其成员包括空闲缓存列表 *fs_free_buffer_list*、正在被使用的缓存列表 *fs_buffer_inuse_list*,以及未被进程引用的缓存列表 *fs_buffer_unused_list*,其中每个缓存单元的取值为{ ⊥, $disk\_block_1$, $disk\_block_2$, …, $disk\_block_n$},即取值或为空、或为磁盘数据块;

3) FS 目录项缓存信息 *fs_dentries*,其成员包括空闲目录项列表 *fs_free_dentry_list*、正在被使用的目录项列表 *fs_dentry_inuse_list*,以及未被进程引用的目录项列表 *fs_dentry_unused_list*. *fs_dentry_unused_list* 中的目录项虽然未被进程引用,但仍然有效,若再次被访问,则无需重新建立,只需移动到 *fs_dentry_inuse_list* 列表;

4) FS 文件索引节点的缓存信息 *fs_inodes*,其成员包括空闲文件索引节点的列表 *fs_free_inode_list*、正在被使用的文件索引节点列表 *fs_inode_inuse_list*,以及未被引用的文件索引节点列表 *fs_inode_unused_list*. 每个文件索引节点的类型是记录类型,定义为:

```
record inode =
    inode_count :: nat
    inode_filelink_count :: nat
    inode_size :: nat
    inode_block_list :: "nat list"
```

成员包括: 进程对文件索引节点的引用计数 *inode_count*、文件链接计数 *inode_filelink_count*、文件大小 *inode_size*,以及文件在磁盘上的数据块数量的计数列表 *inode_block_list*;

5) FS 与系统其它功能模块通信的消息列表 *fs_msgs*. 其中每个消息体的类型是记录类型,成员包括: 消息类型 *m_type*,取值如 *READ*/*WRITE* 等; 文件描述符 *m_fd*; 请求读写的字节数 *m_nbytes*; FS 返回的信息体 *m_reply*.

6) 当前运行进程的 FS 部分描述结构列表 *fs_fprocs*,记录了系统中正在运行进程的 FS 部分的描述信息;

7) 打开文件指针信息列表 *fs_filps*,记录了系统中所有处于打开状态的文件指针对象信息;

8) FS 中的磁盘信息 *fs_disks*,描述了系统中磁盘数据的状态信息,这一部分的内容将在 3.3 节阐述.

3.3　ext3 文件系统形式化模型

FS 设计重要的一个方面是数据文件在物理硬盘上的存放问题. VTOS 采用 ext3 文件系统格式存储数据文件. 为此,有必要说明我们在对 VTOS 文件系统 FS 的设计和验证过程中,如何对物理文件系统构建形式化模型,该模型将应用于后续对 FS 的形式化验证过程中. 为此,本节在上一节对 VTOS FS 的对象抽象及其状态空间的描述基础上,对 VTOS 所采用的 ext3 文件系统建立形式化模型.

一个实际的物理硬盘存储 0/1 字符串,为了描述这些 0/1 字符串所表示的数据文件之间的关系,我们也采用树型(文件目录)结构来抽象数据文件在 ext3 磁盘上的存放,建立磁盘的形式化模型.

首先,我们在 Isabelle/HOL 中定义一个树形(文件目录)结构 file 来抽象地描述 ext3 磁盘上的文件:

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

并给出 *file* 的四个基本操作的定义.

- *getFilename*: 返回指定树节点的值,即文件名;
- *getDir*: 返回以指定树节点为根的子树,即目录结构;
- *isFile*: 以关键字在指定节点的儿子节点中查找文件是否存在,若存在,则返回其值;
- *isDir*: 在树中查找目录是否存在,若存在则返回路径中最后节点的值.

下面,我们根据 ext3 硬盘布局,给出硬盘的形式化定义:

```
record disk =
  fs_super_blocks :: "fs_super_block list"
    ext_grp :: "char list"
      imap :: "(bit list) list"
      zmap :: "(bit list) list"
        files :: "(string, ext3_dir_entry) file"
```

其中,*files* 字段表示硬盘中文件的树形关系,其中树的每个节点有 *string* 和 *ext3_dir_entry* 两个域: *string* 表示文件的名字,*ext3_dir_entry* 则表示与 *string* 所对应文件信息及内容. 树节点的子节点则表示目录文件所包含的文件,树的叶子结点则表示该文件是一个常规文件.

下面的章节对 FS 初始化(*fs_init*)、文件读取(*fs_read*)和文件写入(*fs_write*)服务例程三个部分的操作语义进行定义.

3.4　FS 初始化 *fs_init* 操作语义

在 FS 服务进程启动之后首先进行系统状态的初始化工作,从语义上来说是将系统引导为 FS 工作前所需要的状态,这一部分的任务由 *fs_init* 函数完成. 具体来说,*fs_init* 的操作语义包括: 将磁盘超级块(*super_block*)和 FS 根索引节点(*root inode*)读入内存,并为根索引节点创建相应的目录项缓存 dentry 结构,然后为缓存分配一定数量的空闲空间. *fs_init* 操作语义在 Isabelle/HOL 中定义如下页图 1 所示.

```
fun fs_init:: "state => state" where
"fs_init s =
  s⦇fs_super_blocks := (fs_super_block (fs_disks s)) # ❙,
    fs_inodes := (fs_inodes s)⦇fs_free_inode_list :=
      (fs_free_list_init ❙ (nil_inode) (INODE_NUM -(nat 1))),
      fs_inode_inuse_list :=
      (di_mi (nth (I (fs_disks s)) 2) nil_inode) # ❙,
      fs_inode_unused_list := ❙𝔻,
    fs_dentries := (fs_dentries s)⦇fs_free_dentry_list :=
      (fs_free_list_init ❙ (nil_dentry)
      (DENTRY_NUM -(nat 1))),
      fs_dentry_inuse_list :=
      (i_d (di_mi (nth (I (fs_disks s)) 2) nil_inode) nil_dentry)
      # ❙,
      fs_dentry_unused_list := ❙𝔻,
    fs_buffers := (fs_buffers s)⦇fs_free_buffer_list :=
      (fs_free_list_init ❙ nil_buffer (NR_BUFFER -(nat 1))),
      fs_buffer_inuse_list :=(db_b (CLB (ext_grp (fs_disks s)))
      nil_buffer) # ❙,
      fs_buffer_unused_list := ❙𝔻,
    fs_filps := (fs_free_list_init ❙ nil_filp (NR_FILPS)),
    fs_fprocs := (fs_free_list_init ❙ nil_fproc (NR_FPROCS))
𝔻"
```

图 1 *fs_init* 操作语义定义

Fig. 1 Definition of operation semantics of *fs_init*

3.5 文件读取 *fs_read* 和写入数据 *fs_write* 的操作语义

经过 *fs_init* 的初始化之后，FS 进入等待处理应用程序请求的状态，对于读取文件的服务请求，则执行 *fs_read* 服务例程；对于写入的服务请求，则执行 *fs_write* 服务例程，其操作语义在 Isabelle/HOL 中定义如图 2 和图 3 所示.

```
fun fs_read :: "state => state" where
"fs_read s =( if length (fs_msgs s) =0 then s
else
( if(filp_count (nth (fp_filp (nth (fs_fprocs s) (m_source
(hd (fs_msgs s)))))) (m_fd (hd (fs_msgs s))))) =0
then s
else ( if(( filp_pos (nth (fp_filp (nth (fs_fprocs s)
(m_source (hd (fs_msgs s)))))) (m_fd
(hd (fs_msgs s))))) +
(m_nbytes (hd (fs_msgs s))) > MAX_SIZE)
then s
else (real_read s))))"
```

图 2 *fs_read* 操作语义定义

Fig. 2 Definition of operation semantics of *fs_read*

```
fun fs_write :: "state => state" where
"fs_write s = ( if length (fs_msgs s) =0 then s
else
( if(filp_count (nth (fp_filp (nth (fs_fprocs s) (m_source
(hd (fs_msgs s)))))) (m_fd (hd (fs_msgs s))))) = 0
then s
else ( if(( filp_pos (nth (fp_filp (nth (fs_fprocs s)
(m_source (hd (fs_msgs s)))))) (m_fd
(hd (fs_msgs s))))) +
(m_nbytes (hd (fs_msgs s))) > MAX_SIZE)
then s
else (real_write s))))"
```

图 3 *fs_write* 操作语义定义

Fig. 3 Definition of operation semantics of *fs_write*

其中"*real_read s*"是根据当前系统状态中缓存的使用情况，从不同的链表中申请空闲缓存来暂存用户需要读的数据；"*real_write s*"首先调用"*real_read s*"将需要写到磁盘的数据暂存到缓存 buffer 中，然后根据消息 message 中需要写的数据，调整文件大小，并写入数据.

3.6 VTOS FS 自动机模型

从对象的角度来看，FS 服务的过程就是对对象的改变过程，即系统状态转换的过程. 下面的章节尝试将 FS 以一个状态机的方式来描述.

一个合法的状态转换过程就是由初始状态开始经过一系列的状态转换，最终达到终止状态，这一整个过程中事件的集合即为自动机所接受的输入串.

定义 1. 自动机一般性定义. 确定型自动机是一个五元组:

$M = (Q, \Sigma, \delta, q_0, F)$，其中: Q 是状态集合; Σ 是符号的有限集合，称为输入字母表; δ: $Q \times \Sigma \to Q$ 是一个全函数，称为状态转移函数; $q_0 \in Q$，是初始状态; $F \subseteq Q$，是终止状态的集合.

在定义 1 对自动机一般性定义的基础上，下面我们引出 VTOS FS 自动机模型.

定义 2. VTOS FS 状态机模型. 根据定义 1，建立 FS 状态机模型如下:

1) 状态集合 Q_{FS}

FS 状态如 3.2 节所定义，包括 FS 磁盘超级块信息 *fs_super_blocks*、磁盘数据块在 FS 空间的缓存信息 *fs_buffers*、FS 目录项缓存信息 *fs_dentries*、FS 文件索引节点的缓存信息 *fs_inodes*、FS 与系统其它功能模块通信的消息列表 *fs_msgs*、当前运行进程的文件系统部分描述结构列表 *fs_fprocs*、打开文件指针信息列表 *fs_filps*、FS 中的磁盘信息 *fs_disks*. 同时，我们按照等价类对 FS 状态进行划分，限于篇幅，本文针对 FS 的初始化过程，以及读取 *read* 和写入 *write* 请求服务进行形式化验证，为此 FS 状态分成以下五类，即 $Q_{FS} = S0 \cup S1 \cup S2 \cup S3 \cup S4$，定义如下:

(1) $S0$: 表示在对 FS 初始化之前系统的合法状态集合，即有公式"$\forall s \in S0.\ Valid\_disk(s)$"成立，*Valid_disk* 为磁盘检测函数，表示硬件条件是合法的.

(2) $S1$: 表示在对 FS 初始化之后系统的合法状态集合，满足公式:

"$\forall s \in S1.\ \exists s' \in S0.\ fs\_disks\ s = fs\_disks\ s'$"，

即表示 FS 经过初始化之后，并不改变硬盘数据状态.

(3) $S2$: 表示 FS 接收到读取请求 *read*，准备执行读取服务的状态集合，满足公式:

"$\forall s \in S2.\ m\_type\ (hd\ (fs\_msgs\ s)) = READ$".

(4) $S3$: 表示 FS 接收到写入请求 *write*，准备执行写入服务的状态集合，满足公式:

"$\forall s\ S3.\ m\_type\ (hd\ (fs\_msgs\ s)) = WRITE$".

(5) $S4$: 表示 FS 执行完用户请求，再次可以接受消息进行服务的状态集合，即满足公式:

"$\forall s \in S4.$
$((\exists s' \in S2.\ s = fs\_read\ s') \vee$
$(\exists s' \in S3.\ s = fs\_write\ s'))$
"

2) 符号表 Σ_{FS}

FS 自动机模型可以识别的符号表,包括 FS 提供的服务请求接口以及针对上层应用程序的功能请求所进行的服务例程. 限于篇幅,本文针对 FS 的初始化过程,以及读取 *read* 和写入 *write* 请求服务进行形式化验证,为此本文中 FS 状态机模型的符号表为: $\Sigma_{FS}=\{i, r\_sc, w\_sc, r, w\}$,其中: i 为 FS 初始化过程; r_sc 为提供给上层应用程序的 *read* 系统调用请求; w_sc 为提供给上层应用程序的 *write* 系统调用请求; r 为根据状态中消息类型执行 FS 服务例程 *fs_read*; w 为根据状态中消息类型执行 FS 服务例程 *fs_write*.

3) 状态转移函数 δ_{FS}

在以上对状态集合 Q 和符号表 Σ_{FS} 定义的基础上,对应地,我们将 FS 状态机模型的状态转移函数定义为: $\delta_{FS}=\{\delta_0, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6\}$,对于一个型为 $\delta_i(S, a)=S'$ 的转移函数表示"对于一个属于 S 的 FS 状态 s,存在一个属于 S' 的 FS 状态 s',并且 s' 是由 s 经过动作 a 改变之后的 FS 状态",δ_{FS} 中具体的函数解释如下:

(1) $\delta_0(S0, i)=S1$,表示在系统启动后,FS 可以进行正确的初始化达到统一的状态;

(2) $\delta_1(S1, r\_sc)=S2$,当 FS 接收到 *read* 请求时,将 *read* 消息插入系统状态 *state* 中消息列表 *fs_msgs* 的头部;

(3) $\delta_2(S1, w\_sc)=S3$,当 FS 接收到 *write* 请求时,将 *write* 消息插入系统状态 *state* 中消息列表 *fs_msgs* 的头部;

(4) $\delta_3(S2, r)=S4$,根据系统状态中消息类型,调用服务例程 *fs_read*;

(5) $\delta_4(S3, w)=S4$,根据系统状态中消息类型,调用服务例程 *fs_write*;

(6) $\delta_5(S4, r\_sc)=S2$,FS 在完成处理例程后,再次接收到 *read* 请求,则系统进入 $S2$ 状态;

(7) $\delta_6(S4, w\_sc)=S3$,FS 在完成处理例程后,再次接收到 *write* 请求,则系统进入 $S3$ 状态.

4) 初始状态 q_{0FS}

与上述对状态集合的分析和定义相对应,FS 自动机模型的初始状态满足: $q_{0FS}\in S0$.

5) 终止状态 F_{FS}

在 FS 执行完初始化并进入到接收消息的状态,即上述的状态集 $S1$,我们视其为合法的终止态; 对应的,FS 处理完消息后的状态 $S4$,同样视为可终止的系统状态. 为此终止态 $F_{FS}=S1\cup S4$.

综上所述,FS 的状态机可以表示为这样的五元组: $M_{FS}=(Q_{FS}, \Sigma_{FS}, \delta_{FS}, q_{0FS}, F_{FS})$.

4　VTOS 文件系统正确性的形式化验证

本节在第 3 节给出的 VTOS FS 自动机模型和形式化分析的基础上,尝试给出 FS 的正确性断言,并借助 Isabelle/HOL 对其进行推理验证. 限于篇幅,本章针对 FS 的初始化过程,以及读取 *read* 和写入 *write* 请求服务的正确性进行形式化验证.

4.1　FS 初始化 *fs_init* 的正确性验证

在 VTOS 的设计过程中,我们认为 FS 初始化的正确性是指,给定一个正常的可识别的磁盘,*fs_init* 能将其初始化为特定的初始状态,即 M_{FS} 中的 S_1 状态.

定理 1. FS 初始化正确性.

Theorem *fs_init_correctness*:

$$\begin{aligned}&"\ \forall s\in S0.\ (Valid\_disk\ (fs\_disks\ s)\rightarrow\\&\quad Valid\_disk\ (fs\_disks\ (fs\_init\ s))\ \wedge\\&\quad Valid\_buffer\ (fs\_init\ s)\ \wedge\\&\quad Valid\_dentry\ (fs\_init\ s)\ \wedge\\&\quad Valid\_inode\ (fs\_init\ s))"\end{aligned}\tag{1}$$

定理 1 表明对于 $S0$ 中的状态 s,如果满足磁盘检测(*Valid_disk* (*fs_disks s*)),即硬件条件是合法的,那么经过 FS 初始化(*fs_init s*) 后的状态满足"磁盘数据块在 FS 空间的缓存信息"正确性检测(*Valid_buffer* (*fs_init s*))、"FS 目录项缓存信息"正确性检测(*Valid_dentry* (*fs_init s*)) 和"FS 文件索引节点的缓存信息"正确性检测(*Valid_inode* (*fs_init s*)).

4.2　fs_read 服务例程正确性验证

对于 VTOS 文件系统中 read 系统调用功能的正确性,即服务例程 fs_read 的功能语义正确性,我们定义为: "假设在系统状态 s 下,系统接收到上层应用程序的 read 请求消息,如果状态 s 满足 Valid_disk、Valid_buffer、Valid_inode、Valid_dentry 判定条件,那么在执行完 fs_read 服务例程后上述四个判定条件依然成立",即如下面的定理表述:

定理 2. *fs_read* 服务例程初始化正确性.

Theorem *fs_read_correctness*:

$$\begin{aligned}&"\ \forall s\in S1.\ (Valid\_disk\ (fs\_disks\ s)\ \wedge\\&\qquad m\_type\ (hd\ (fs\_msgs\ s))\ =\ READ\rightarrow\\&\quad Valid\_disk\ (fs\_disks\ (fs\_read\ s))\ \wedge\\&\quad Valid\_buffer\ (fs\_read\ s)\ \wedge\\&\quad Valid\_dentry\ (fs\_read\ s)\ \wedge\\&\quad Valid\_inode\ (fs\_read\ s))"\end{aligned}\tag{2}$$

对于上述的 fs_read 正确性定理,由于在 fs_read 对数据对象的操作过程,即功能语义中,涉及到对上层程序需要读取的数据在缓存中的存取操作,即 3.5 节阐述的 real_read 操作,为此需要首先对 real_read 关于上述四个判定条件进行验证,如下所示:

Lemma *real_read_correctness*:

$$\begin{aligned}&"\ \forall s\in S1.\ (Valid\_disk\ (fs\_disks\ s)\ \wedge\\&\qquad Valid\_buffer\ s\ \wedge\\&\qquad Valid\_dentry\ s\ \wedge\\&\qquad Valid\_inode\ s\rightarrow\\&Valid\_disk\ (fs\_disks\ (real\_read\ s))\ \wedge\\&Valid\_buffer\ (real\_read\ s)\ \wedge\\&Valid\_dentry\ (real\_read\ s)\ \wedge\\&Valid\_inode\ (real\_read\ s))"\end{aligned}\tag{3}$$

4.3　*fs_write* 服务例程正确性验证

与 *fs_read* 服务例程的正确性验证类似,我们有如下的正确性定理:

定理 3. *fs_write* 服务例程初始化正确性.

Theorem *fs_write_correctness*:

$$\begin{aligned}&"\ \forall s\in S1.\ (Valid\_disk\ (fs\_disks\ s)\ \wedge\\&\qquad m\_type\ (hd\ (fs\_msgs\ s))\ =\ WRITE\rightarrow\\&\quad Valid\_disk\ (fs\_disks\ (fs\_write\ s))\ \wedge\end{aligned}$$

Valid_buffer (*fs_write s*) ∧
Valid_dentry (*fs_write s*) ∧
Valid_inode (*fs_write s*)) " (4)

对于上述的 *fs_write* 正确性定理,同样涉及到对上层程序需要写入的数据在缓存中的存取操作,即 3.5 节阐述的 *real_write* 操作,为此需要首先对 *real_write* 关于上述四个判定条件进行验证,如下所示:

Lemma *real_write_correctness*:

" ∀ *s* ∈ *S*1. (*Valid_disk* (*fs_disks s*) ∧
Valid_buffer s ∧
Valid_dentry s ∧
Valid_inode s→
Valid_disk (*fs_disks* (*real_write s*)) ∧
Valid_buffer (*real_write s*) ∧
Valid_dentry (*real_write s*) ∧
Valid_inode (*real_write s*)) " (5)

我们的验证环境采用 Dell Studio XPS 9100 平台,处理器为 2.8GHz Intel i7 930,内存为 4GB,OS 采用 Ubuntu LTS 12.04,Isabelle 采用 Isabelle2013 版本. VTOS 文件系统的整个 Isabelle 验证工程代码量大概在 4.6k SLOC 左右.

5 结束语

本文采用形式化方法对微内核架构 FS 的设计和实现过程进行了形式化的描述和验证,建立了 FS 的状态自动机模型,在此基础上描述 FS 初始化过程、系统服务例程的功能语义和系统状态转化函数,并对 FS 的正确性问题进行了验证. FS 作为系统的一个功能模块,其运行过程和其他的模块如进程管理、内存管理存在很多交互的环节,如何描述和验证 FS 与其他模块的整合的正确性是我们未来努力的方向.

References:

[1] Morgan Carroll, Sufrin Bernard. Specification of the UNIX filing system [J]. IEEE Transactions on Software Engineering, 1984, SE-10(2): 128-142.

[2] Wenzel Markus, München TU. Some aspect of UNIX file-system security [C]. In: München TU ed. Isabelle/Isar Proof Document, 2001.

[3] Abrial Jean-raymond. Modelling in Event-B: system and software engineering [M]. Cambridge: Cambridge University Press, 2008.

[4] Coleman Joey, Jones Cliff, Oliver Ian, et al. RODIN(Rigorous open development environment for complex systems) [C]. Proceedings of 5th European Dependable Computing Conference, 2005: 23-26.

[5] Damchoom Kriangsak, Butler Michael, Abrial Jean-raymond. Modelling and proof of a tree-structured file system in Event-B and rodin [C]. Proceedings of International Conference on Formel Engineering Methods, 2008: 25-44.

[6] Owre Sam, Rushby John, Shankar Natarajan. PVS: a prototype verification system [C]. Proceedings of the 11th Internationl Conference on Aufomated Deduction, 1992: 748-752.

[7] Hesselink Wim, Lali Muhammad. Formalizing a hierarchical file system [J]. Formal Aspects of Computing, 2012, 24(1): 27-44.

[8] Holzmann Gerard. The SPIN model checker [M]. Boston: Addison-Wesley, 2003.

[9] Ciardo Gianfranco, Jones Robert-lnzey, Miner Andrew-stephen, et al. Logic and stochastic modeling with SMART [J]. Performance Evaluation, 2006, 63(6): 578-608.

[10] Galloway Andy, Lüttgen Gerald, Mühlberg Jan-tobias, et al. Model-checking the linux virtual file system [C]. Proceedings of International Conference on Verification, Model Checking, and Abstract Interpretation, 2009: 74-88.

[11] Nipkow Tobias, Paulson Lawrence. Isabelle/HOL: a proof assistant for higher-order logic [M]. Berlin: Springer, 2002.

[12] Qian Zhen-jiang, Liu Wei, Huang Hao. OSOSM: operating system object semantics model and formal verification [J]. Journal of Computer Research and Development, 2012, 49(12): 2702-2712.

附中文参考文献:

[12] 钱振江,刘 苇,黄 皓. 操作系统对象语义模型(OSOSM)及形式化验证 [J]. 计算机研究与发展,2012,49(12):2702-2712.