微内核架构内存管理的形式化设计和验证方法研究

钱振江[1,2],刘永俊[2],姚宇峰[2],汤 力[2],黄 皓[1],宋方敏[1]
(1. 南京大学计算机科学与技术系,江苏南京 210023;2. 常熟理工学院计算机科学与工程学院,江苏苏州 215500)

摘 要: 由于巨大的规模和复杂性,操作系统的设计和实现的正确性很难用传统的定量方法来描述. 本文阐述对微内核操作系统的形式化设计和验证的方法. 在汇编层利用非确定性自动机对系统进行形式化建模,并使用 Hoare 三元组描述模块接口函数的前后置条件,作为函数正确性的定义. 以实现的 VSOS(Verified Secure Operating System)内存管理模块为例,在 Isabelle/HOL 定理证明器环境中对建立的内存管理模型和系统行为的操作语义进行形式化描述,并对内存管理模块的设计和实现的正确性进行验证. 结果表明,这一方法是可行的和高效的.

关键词: 操作系统; 内存管理; 形式化设计; 形式化验证; 定理证明
中图分类号: TP316 **文献标识码:** A **文章编号:** 0372-2112 (2017)01-0251-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.01.035

Research on Method of Formal Design and Verification of Memory Management Based on Microkernel Architecture

QIAN Zhen-jiang[1,2], LIU Yong-jun[2], YAO Yu-feng[2], TANG Li[2], HUANG Hao[1], SONG Fang-min[1]
(1. *Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu* 210023, *China*;
2. *School of Computer Science and Engineering, Changshu Institute of Technology, Suzhou, Jiangsu* 215500, *China*)

Abstract: The correctness of design and implementation of operating systems is difficult to be described with the traditional quantitative methods, because of the enormous size and complexity. This paper illustrates our method of formal design and verification of microkernel operating system. We construct the system model with the non-deterministic automaton in the assembly level, and use the Hoare triples to describe the previous and post conditions of the interface functions, as the definitions of function correctness. We take the module of memory management in the self-implemented VSOS (Verified Secure Operating System) as an example, to illustrate our formal method. Meanwhile, we formally describe the constructed memory management model and operation semantics of system behaviors in the Isabelle/HOL theorem prover, and verify the correctness of design and implementation of the memory management. The result shows that the method proposed is feasible and efficient.

Key words: operating system; memory management; formal design; formal verification; theorem proving

1 引言

对于关键的核心软件系统如操作系统(Operating System,OS),如何保证其正确可靠一直以来都是学术界和工业界努力的方向. 形式化验证技术以数理逻辑为基础,具有严谨性的特点,目前被公认为是能够保证软件正确性的关键技术. 形式化验证是指根据系统的规范或属性,使用形式化方法验证其正确性或非正确.

Verisoft 是一个大型的计算机系统设计开发项目[1],目标在于对整个计算机系统自底向上从硬件层到应用软件层进行普适形式化证明(Pervasive Formal Verification). 由于系统的庞大规模,内核部分验证所花费的人力和物力是巨大的,并且除内核外的其他模块并未得到完全验证[2].

从上世纪 90 年代开始到现在,耶鲁大学 Shao Zhong 教授带领的 Flint 项目组在形式化验证方面做了大量的工作[3],实现了一种新的逻辑编程语言 VeriML[4],提出的 λHOL^{ind} 逻辑加入了对数据类型的归纳定义,提供了丰富的形式化描述能力以及类型安全特性. 同时,Flint 项目还研究了针对并行程序的验证方法[5],以及采用分层抽象的方法验证 OS 中的各种功能模块[6].

收稿日期:2015-05-15;修回日期:2015-08-17;责任编辑:覃怀银
基金项目:国家自然科学基金(No. 61402057);江苏省科技计划自然科学研究项目(No. BK20140418);中国博士后科学基金(No. 2015M571737);江苏省"六大人才高峰"高层次人才项目(No. 2011-DZXX-035);江苏省高校自然科学研究项目(No. 12KJB520001)

由澳大利亚国家 ICT 实验室(NICTA)主导的 L4.verified 项目旨在对 seL4 微内核进行机械化的正确性验证. Klein 在报告[7]中介绍了验证的工作. 验证的目标是使用 Isabelle/HOL[8]证明最终的执行效果符合抽象模型的预期定义. L4. verified 项目主要集中于对单核版本的 OS 系统进行形式化验证,并未过多地考虑对多核环境的正确性验证.

在之前的 OS 形式化设计和验证工作中,本项目组提出了采用 OS 对象语义模型(OSOSM)[9]对 OS 的操作语义进行描述和验证的方法,并实现了安全可信 OS 原型 VTOS(Verified Trusted Operating System). 本文以实现的 VTOS 后继版本 VSOS(Verified Secure Operating System)的内存管理模块为例,阐述对 OS 的形式化设计和验证的方法. 在考虑并发因素的前提下,利用非确定性自动机[10]在汇编层对 VSOS 的内存管理模块进行形式化建模,并使用 Hoare 三元组描述模块接口函数的前后置条件,作为函数正确性的定义. 同时,本文对建立的内存管理模型和系统的操作语义在 Isabelle/HOL 定理证明器中进行描述,尝试在 Isabelle/HOL 环境中对 VSOS 内存管理模块的设计和实现的正确性进行验证.

2 VSOS 内存管理模块的设计

2.1 VSOS 的整体框架

VSOS 的整体框架采用层次化结构,共分成 4 层. 最底层是运行于核心态的代码,由微内核、系统任务和时钟任务组成,它们共享地址空间. 微内核运行在特权级 0,负责处理中断和异常、进程调度以及进程间通信(即消息的发送与接收);系统任务运行在特权级 1,负责向用户态的系统进程提供内核调用的服务,这些内核调用包括读写 I/O 端口、在进程间拷贝数据等,它们是一组特权的调用,仅授权的用户态系统进程能够请求这组服务,其他的用户进程则无法请求这组服务;时钟任务运行在特权级 1,负责管理硬件时钟,用户态进程无法直接获取它的服务,而必须通过系统任务间接地获取它的服务.

位于第 2 层的是设备驱动,包括终端管理和磁盘管理等. 直接向用户进程提供服务的是位于第 3 层的服务器. 这些服务器提供的服务即作为 OS 向用户提供的接口,称作系统调用. 第 4 层包括 Shell 以及用户程序,其中 Shell 是用户与 OS 的交互接口.

2.2 内存管理模块的设计

VSOS 将内存管理模块置于系统任务中,通过内核调用接口向用户态系统进程提供服务. 将内存管理模块置于系统任务中,可以在基本不破坏微内核结构的情况下,减少进程间消息通信的次数,从而提高系统的性能.

内存管理模块的任务本质是管理地址转换,并且实现细节对上层用户进程来讲是透明的,向用户进程提供一个抽象的、一致的虚拟地址空间视图. VSOS 的内存管理模块结构按照功能分解被划分为两个子模块:页框分配器和进程虚存管理. 页框分配器子模块负责管理系统的物理页框资源,向上提供分配页框(alloc_pages)及释放页框(free_pages)的接口函数. 进程虚存管理子模块在页框分配器的基础之上负责管理进程的虚拟地址空间,向上提供创建虚拟地址区间(mmap)和删除虚拟地址区间(munmap)、以及扩展用于分配动态内存的堆区(brk)的接口函数,为用户进程提供一个一致的、抽象的虚拟地址空间视图.

3 VSOS 内存管理模块的形式化建模

本节以 VSOS 内存管理模块中页框分配器子模块的功能为例,阐述 VSOS 内存管理模块的形式化建模方法,借助 Isabelle/HOL 定理证明器从状态、字母表、函数的操作语义、状态转换函数,和函数操作的前后置条件等方面给出模型的形式化描述.

(1)状态

页框分配器的状态及相关常量在 Isabelle/HOL 中的定义如下.

MaxPage 表示最大的物理页框号,值暂定为 0x7fff,表明物理内存总大小,MaxPage 的值可根据实际情况修改;MaxOrder 表示空闲链表的最大级数,值为 10,表明支持的最大空闲块为 2^{10} 个页;U 是物理页框构成的全集,{0…MaxPage}是 Isabelle/HOL 中的集合表示方法,表示由大于等于 0 且小于等于 MaxPage 的整数构成的集合.

pa_state 表示页框分配器的状态,类型为 record,其中字段 Free 是一个从 nat 到"int list"的映射,给定输入参数 i($i \leqslant$MaxOrder),函数值"Free i"表示 i 级空闲链表中所有空闲块的编号;字段 Allocated 用来表示已分配物理页框的编号所构成的集合. order、num、retval 等字段用来记录参数和返回值.

state 用来表示机器状态,其中字段 C 是从地址到指令的函数映射,用来表示程序指令段,"word32"表示 32 比特的字类型;字段 D 是从地址到字节类型的映射,用来表示程序指令所操作的数据;字段 R 是对寄存器组的建模,其类型为构造的新类型 regs,包含有通用寄存器、指令指针、标志寄存器和段寄存器;call_ret 字段用来记录函数调用的深度,每执行一次函数调用指令 call 时,call_ret 字段加 1,执行函数返回指令 ret 时,call_ret 字段减 1.

(2)字母表

页框分配器的字母表在 Isabelle/HOL 中使用 data-

type 定义为 pa_Ops,即页框分配器的操作列表,包含操作子 alloc_pages 和 free_pages. 其中"alloc_pages nat"为分配页框操作,nat 类型参数指出分配阶 order,即请求分配 2^{order} 个页;"free_pages int nat"为释放页框操作,int 类型参数指出要释放的页框块的首页框号(num),nat 型参数 order 指出释放从编号 num 开始的 2^{order} 个页.

(3)函数的操作语义

为了在定理证明器 Isabelle/HOL 中验证函数的正确性,将函数的操作语义在 Isabelle/HOL 中描述.

限于篇幅,在系统状态和指令类型的基础上,本文以页框分配函数 alloc_pages 为例阐述对于函数的操作语义的定义方法.

与 alloc_pages 函数的汇编代码相对应地,alloc_pages 函数在 Isabelle/HOL 中的操作语义以函数运行的部分前置条件的形式来定义,如算法 1 所示.

算法 1　alloc_pages **操作语义在** Isabelle/HOL **中的定义**

```
1  definition P0::"state => bool" where
2  "P0 s≡(
3  ((C s) 0xc01015a3 = pushl 1 push_r 0 ebp eiz 0) ∧
4  ((C s) 0xc01015a4 = movl 2 rr 0 esp eiz 0 0 ebp eiz 0) ∧
5  ((C s) 0xc01015a6 = subl 3 ir 0x2c eiz eiz 0 0 esp eiz 0) ∧
6  ((C s) 0xc01015a9 = cmpl 4 im 0xa eiz eiz 0 8 ebp eiz 0) ∧
7  ……
8  ((C s) 0xc010166c = leave 1) ∧
9  ((C s) 0xc010166d = ret 1) ∧
10 (EIP (R s) = 0xc01015a3) "
```

(4)状态转换函数

本文定义 VSOS 形式化模型的状态转换函数为 next,"next s i"表示在给定的状态 s 下执行指令 i 之后的状态. 定义 step 函数根据当前状态下的指令指针指向的指令来运行从而改变机器的状态.

(5)函数操作的前后置条件

本小节首先给出 VSOS 内存管理模块的不变式的描述,在此基础上给出函数操作的前、后置条件的描述.

对于页框分配器子模块的不变式,主要有:(1)空闲页框集与已分配页框集不相交,记为 pa_Inv1;(2)空闲页框集与已分配页框集的并集等于物理页框所构成的全集,记为 pa_Inv2;(3)不同级别链表上的空闲页框集不相交,记为 pa_Inv3;(4)i 级($i \leqslant$ MaxOrder)链表上空闲页框块的起始页编号能够被 2^i 整除,记为 pa_Inv4;(5)伙伴页框块[11]不可能同在 i 级链表中(最高级除外),记为 pa_Inv5. 便于表达,本文使用 pa_Inv 表示 pa_Inv1 ~ pa_Inv5 的合取.

在页框分配器的不变式定义的基础上,本文以页框分配函数 alloc_pages 为例阐述函数前后置条件的描述方法.

假设页框分配函数 alloc_pages 将自动机的状态从 s 转变为 s'. 下面章节分两种情况来描述 alloc_pages 操作的前后置条件.

(a)输入参数合法

alloc_pages 的函数形式中包含输入参数 order,标示要分配 2^{order} 个连续物理页. 如果 order 合法,即 order 所指定的要分配的物理页的阶在页框分配器所支持的范围之内. 前置条件记为 precon1_alloc.

输入参数合法时,分配操作既可能成功也可能失败. 下面继续分情况说明:

分配失败:这是由于系统没有足够的页框可供分配,或者刚开始时有足够的页框,但由于并发的影响,到了实际分配时空闲页却不够了. 在这种情况下,返回值 retval 为 -1,并且系统状态需满足不变式 pa_Inv;

分配成功:此时返回值 retval 是 2^{order} 的整数倍,并且分配出的页框块属于已分配物理页框集合 Allocated.

综合上述两种情况,输入参数合法时,分配操作 alloc_pages 的后置条件记为 postcon1_alloc.

(b) 输入参数非法

输入参数 order 非法,即 order 所指定的要分配的物理页的阶不在页框分配器所支持的范围之内. 此时前置条件记为 precon2_alloc.

由于输入参数非法,分配操作必然失败. 此时后置条件记为 postcon2_alloc.

总结(a)、(b)两种情况,alloc_pages 操作的语义用 Hoare 三元组描述如下:

$$\{\text{precon1\_alloc}\}\ \text{alloc\_pages}\ \{\text{postcon1\_alloc}\}$$
$$\|\ \{\text{precon2\_alloc}\}\ \text{alloc\_pages}\ \{\text{postcon2\_alloc}\}$$

4　VSOS 内存管理模块的形式化验证

4.1　证明思路

为了便于阐述和理解,本小节用半形式化的数学语言在 C 语言级来阐述证明的思路和框架. 页框分配函数 alloc_pages 的 C 语言实现如算法 2 所示,此处以页框分配函数 alloc_pages 参数合法时的情况进行说明.

算法 2　alloc_pages **函数的 C 语言代码**

```
1  int alloc_pages(unsigned int order)
2  {free_area_t *area;
3  unsigned int cur_order;
4  struct list_head *head;
5  struct list_head *cur;
6  page_t *page;
7  int index;
```

```
8   if(order > MAX_ORDER) return -1;
9   cur_order = order;
10   area = mem_zone.free_area + order;
11   head = &area ->free_list;
12   cur = head ->next;
13   while((cur_order < MAX_ORDER) && (cur = =head))
14   {cur_order + +;
15     area + +;
16     head = &area ->free_list;
17     cur = head ->next;}
18   if(cur = =head) index = -1;
19   else
20   {cur ->next ->prev = cur ->prev;
21     cur ->prev ->next = cur ->next;
22     page = ((page_t *)((char *)(cur) -
         (unsigned long)(&((page_t *)0) ->buddy_list)));
23     index = page - mem_map;
24     split(page,order,cur_order);}
25   return index;}
```

验证过程将 alloc_pages 函数划分为 B_1、B_2 两个部分，B_1 对应算法 1 中的 2～11 行，B_2 对应 12～25 行. B_1 是非临界区，仅访问栈上的局部变量，其所对应的汇编代码中的每条指令是一个原子块，不会被干扰，但指令之间可能被其他干扰进程插入执行. 这里将其他修改页框分配器全局变量的进程称为干扰进程.

B_2 会改变页框分配器的全局变量，属于临界区，因此需要作为一个原子块，通过互斥机制来保护其免受并发的干扰.

验证过程先分别论证 B_1、B_2 的正确性，然后将两者结果综合起来得到函数的正确性证明. 记开始执行 B_1 时的状态为 s_0，B_1 执行结束时的状态为 s_k. 记 B_2 开始执行时的状态为 s_a，由于并发机制的存在，s_a 可能不同于 s_k. B_2 中循环语句开始执行时的状态为 s_b，循环语句执行结束时的状态为 s_e，B_2 执行结束时的状态为 s_n.

(1) B_1 的正确性

在参数合法的情况下，s_0 应该满足第 3 节中所定义的相应前置条件，即 s_0 满足：

$$s_0.\text{order} \leqslant \text{MaxOrder} \tag{1}$$

因此可知 B_1 中分支语句的条件不成立. 虽然 B_1 执行过程中可能有干扰进程执行，但干扰进程仅改变全局变量，不会改变 alloc_pages 的栈上变量. 可以证明，对于执行完 B_1 的状态 s_k，栈上的局部变量被正确设置，即满足如下条件：

$$\text{cur\_order} = s_0.\text{order} \tag{2}$$

$$\text{area} = \text{free\_area} + s_0.\text{order} \tag{3}$$

$$\text{head} = \&(\text{area} \rightarrow \text{free\_list}) \tag{4}$$

其中 cur_order 是当前查找的空闲链表级数；area 用于定位当前级别链表；head 指向当前链表的头结点.

(2) B_2 的正确性

循环前仅有一条语句，即第 12 行的语句. 该语句将局部变量 cur 设置为：

$$\text{cur} = \text{head} \rightarrow \text{next} \tag{5}$$

本文借助于循环不变式来证明其正确性，总结 B_2 中循环语句的循环不变式如下：

$$\text{area} = \text{free\_area} + \text{cur\_order} \tag{6}$$

$$\text{head} = \&(\text{area} \rightarrow \text{free\_list}) \tag{7}$$

$$\text{cur} = \text{head} \rightarrow \text{next} \tag{8}$$

$$F_i = \phi(\text{order} \leqslant i < \text{cur\_order}) \tag{9}$$

式(6)～(8)是关于栈上局部变量的. 式(9)是关于全局变量的，其中 F_i 表示 i 级空闲链表中的空闲块构成的集合，即页框分配器状态 pa_state 中的字段 Free.

对于循环语句开始执行时的状态 s_b，其栈上的变量仍然满足式(2)～(5). 可知 s_0.order = cur_order，不难得出 s_b 满足式(6)～(8).

因 cur_order = s_0.order，不存在 i 使得 s_0.order $\leqslant i <$ cur_order，所以式(9)也满足. 因此 s_b 满足循环不变式.

证明每次循环都能维持不变式，也就是证明在循环条件(记为 B)和循环不变式(记为 P)满足的前提下，执行一次循环(记为 C)后，循环不变式 P 仍满足，即 $\{B \wedge P\}\, C\, \{P\}$.

B_2 中的循环语句有两个控制循环的子条件：当前级别 cur_order 小于空闲链表的最大级数 MaxOrder(记为 p)，当前级别链表为空(记为 q)，所以循环条件 $B = p \wedge q$.

假设 B 成立，即 p 与 q 均为真；并且当前循环开始时循环不变式满足，即式(6)～(9)成立. 因为 q 为真，可得当前链表为空：$F_{\text{cur\_order}} = \phi$. 而 14～17 行的循环体将当前链表定位到下一级：cur_order 增加 1，area 指向下一级，head 和 cur 也被相应更新. 因此当前循环结束时，式(6)～(9)仍然成立，即每次循环能够维持循环不变式.

根据前置条件式(1)得初始时 s_0.order $\leqslant$ MaxOrder，而每次循环都将 order 增加 1，为此必然会导致某次循环开始时 p 不成立，所以循环会终止.

为此，对于循环终止时的状态 s_e：循环条件 B 不成立，但满足式(6)～(9)，即循环不变式 P 仍满足.

循环结束时，B 不成立，可分为三种情况：p 不成立而 q 成立，p 不成立并且 q 不成立，以及 p 成立但 q 不成立.

如果 p 不成立而 q 成立，此时 cur_order = MaxOrder，$F_{\text{cur\_order}} = \phi$，即无足够空闲页框可供分配，18 行 if 语句的条件成立，返回值 index 被置为 -1. 因此状态 s_n 相对于状态 s_a 的改变是：

$$s_n.\ \text{retval} = -1 \tag{10}$$

$$s_n.\ F_i = s_a.\ F_i\ (0 \leqslant i \leqslant \text{MaxOrder}) \tag{11}$$

$$s_n.\ \text{Allocated} = s_a.\ \text{Allocated} \tag{12}$$

如果 p 不成立并且 q 不成立,或者 p 成立而 q 不成立,此时 order ⩽ cur_order ⩽ MaxOrder, $F_{\text{cur\_order}} \neq \phi$,即可从 cur_order 级分配页框块给请求者. 18 行 if 语句的条件不成立,20、21 行从 cur_order 级链表中取下第一个空闲块,22、23 行计算空闲块的首页框号,24 行的函数调用将页框块的多余部分分割插入到低级别的 order ~ cur_order − 1 级链表中. B_2 执行结束时的状态 s_n 相对于状态 s_a 的改变是:

$$s_n.\ \text{retval} = x \tag{13}$$

$$s_n.\ F_{\text{cur\_order}} = s_a.\ F_{\text{cur\_order}} - \text{FB}_x^{\text{cur\_order}} \tag{14}$$

$$s_n.\ F_i = s_a.\ F_i \cup\ \text{FB}_{x_i}^{i}\ (\text{order} \leqslant i < \text{cur\_order}) \tag{15}$$

$$s_n.\ \text{Allocated} = s_a.\ \text{Allocated} \cup \text{FB}_x^{\text{order}} \tag{16}$$

其中,x 表示从 cur_order 链表中移除的页框块(大小为 $2^{\text{cur\_order}}$)的首页框号;$x_i = x + 2^i$,是从所移除页框块中分割出来的插入到 i 级链表中的页框块(大小为 2^i)的首页框号. 其中,$\text{FB}_m^n = \{p_m, p_{m+1}, \cdots, p_{m+2^n-1}\}$ 表示首页框号为 m,大小为 2^n 的页框块.

(3)函数的正确性

综合(1)、(2),B_2 执行结束时的状态 s_n 有两种情形. 当 p 不成立而 q 成立时,s_n 相对于 s_a 的状态改变如式(10) ~ (12)所示,得出 s_n 满足:

$$s_n.\ \text{retval} = -1 \wedge \text{pa\_Inv}(s_n) \tag{17}$$

如果 p 不成立并且 q 不成立,或者 p 成立而 q 不成立,则 s_n 相对于 s_a 的状态改变如式(13) ~ (16)所示. 根据 pa_Inv4 的定义可知:$x \% 2^{\text{cur\_order}} = 0$,即 $s_n.\ \text{retval} \% 2^{\text{cur\_order}} = 0$,并且 $s_n.\ \text{retval} \% 2^{\text{order}} = 0$.

由式(16)可得:

$$\{p_{s_n.\ \text{retval}}, \cdots, p_{(s_n.\ \text{retval}+2^{\text{order}}-1)}\} \subseteq s_n.\ \text{Allocated} \tag{18}$$

根据式(14) ~ (16)得:

$$s_n.\ F = S_a.\ F - \text{FB}_x^{\text{order}} \tag{19}$$

又 s_a 满足 pa_Inv1 和 pa_Inv2,所以 s_n 仍满足 pa_Inv1、pa_Inv2.

由 $x_i = x + 2^i$ 及 $x \% 2^{\text{cur\_order}} = 0$ 得,页框块 $\text{FB}_x^{\text{order}}$、$\text{FB}_{x_i}^{i}$ (order ⩽ i < cur_order)的并集为 $\text{FB}_x^{\text{cur\_order}}$,且两两互不相交,且 $x_i \% 2^i = 0$. 根据式(14)、(15),以及 s_a 满足 pa_Inv3、pa_Inv4、pa_Inv5,可得 s_n 仍满足 pa_Inv3、pa_Inv4、pa_Inv5,即"空闲链表不相交;i 级空闲链表中空闲块首页框号能被 2^i 整除;一对伙伴页框块不可能同在 i 级数(最高级除外)链表中".

所以当 p 不成立并且 q 不成立,或者 p 成立而 q 不成立时,s_n 满足:

$$\begin{aligned}&(s_n.\ \text{retval} \% 2^{\text{order}} = 0) \wedge \\ &\{p_{s_n.\ \text{retval}}, \cdots, p_{(s_n.\ \text{retval}+2^{\text{order}}-1)}\} \subseteq s_n.\ \text{Allocated} \wedge \\ &\text{pa\_Inv}(s_n)\end{aligned} \tag{20}$$

综合式(17)和(20),函数执行结束时的状态 s_n 满足所定义的后置条件.

对于页框分配函数 alloc_pages 参数非法时的情况,可以按照上述的情况进行分析.

4.2 Isabelle/HOL 中的形式化验证

在第 3 节对 VSOS 内存管理模块的形式化模型的描述和 4.1 节对证明过程的思路阐述的基础上,本节以页框分配函数 alloc_pages 为例在 Isabelle/HOL 中验证其正确性.

根据第 3 节的分析,本节对 alloc_pages 的正确性验证分成参数合法和非法两种情况分别进行验证.

定理 1　页框分配函数 alloc_pages 的参数合法情况下的正确性　对于参数合法的情况,alloc_pages 的正确性定理如下:

```
theorem  alloc_correctness_1:
(P0 s0) ∧ (precon1_alloc s0) ∧
(sn = stepn (s0, alloc_pages))
⇒postcon1_alloc s0 sn
```

定理表明:对任意给定的状态 s_0,在其满足前置条件 precon1_alloc 以及 $P0$ 的条件下,执行 alloc_pages 函数后,状态转换为 s_n,s_0 和 s_n 满足后置条件 postcon1_alloc,stepn (s_0, alloc_pages)表示在状态 s_0 下多步执行 alloc_pages 函数体后的状态.

按照 4.1 节的分析,alloc_pages 函数在汇编级划分成与 C 语言级 B_1、B_2 对应的两个部分,对 B_1、B_2 正确性这两个子目标分别进行验证.

对于内存管理模块的其他函数,其验证方法也采用上述的方法. VSOS 内存管理模块在 Isabelle/HOL 中的验证代码量在 14k SLOC(Source Lines of Code)左右,其实现部分的汇编代码量在 1.7k SLOC.

5　结论

本文阐述了微内核 OS 的形式化设计和验证方法. 以实现的微内核 VSOS 内存管理模块为例,在汇编层利用非确定性自动机进行形式化建模,并使用 Hoare 三元组描述模块接口函数的前后置条件,作为函数正确性的定义. 在 Isabelle/HOL 定理证明器环境中对建立的内存管理模型和系统行为的操作语义进行形式化描述,并对 VSOS 内存管理模块的设计和实现的正确性进行验证.

对于 VSOS 的完整验证,工作量共耗费了 18 人年,不可否认,这是一个非常耗时的研究工作. 为此,如何高效地采用模块化的验证策略,以及验证的复用问题是

接下来工作的主要方向.

参考文献

[1] Alkassar E, Hillebrand M A, Leinenbach D, et al. The Verisoft approach to systems verification[A]. Proceedings of the 2nd Working Conference on Verified Software: Theories, Tools, and Experiments[C]. Heidelberg: Springer – Verlag, 2008. 209 – 224.

[2] Leinenbach D, Petrova E. Pervasive compiler verification — from verified programs to verified systems[A]. Proceedings of the 3rd International Workshop on Systems Software Verification [C]. Amsterdam, Holland: Elsevier, 2008. 23 – 40.

[3] Shao Z. Certified software[J]. Communications of the ACM, 2010, 53(12): 56 – 66.

[4] Stampoulis A. VeriML: A Dependently-Typed, User-Extensible, and Language-Centric Approach to Proof Assistant [D]. New Haven: Yale University, 2012.

[5] Liang H, Feng X, Shao Z. Compositional verification of termination-preserving refinement of concurrent programs [A]. Proceedings of the 23rd EACSL Annual Conference on Computer Science Logic and 29th Annual IEEE Symposium on Logic in Computer Science (CSL-LICS'14)[C]. Vienna, Austria: ACM, 2014. 65: 1 – 65: 10.

[6] Carbonneaux Q, Hoffmann J, Ramananandro T, Shao Z. End-to-end verification of stack-space bounds for C programs[A]. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation[C]. Edinburgh, UK: ACM, 2014. 270 – 281.

[7] Klein G, Andronick J, Elphinstone K, Murray T, Sewell T, Kolanski R, Heiser G. Comprehensive formal verification of an OS microkernel[J]. ACM Transactions on Computer Systems, 2014, 32(1): 1 – 70.

[8] Nipkow T, Paulson L C, Wenzel M T. Isabelle/HOL: A Proof Assistant for Higher-Order Logic[M]. Heidelberg: Springer, 2002.

[9] 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.
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. (in Chinese)

[10] Sipser M. Introduction to the Theory of Computation [M]. Boston: Cengage Learning, 2012.

[11] Knuth D E. The Art of Computer Programming, Volume 1: Fundamental Algorithms[M]. Boston: Addison-Wesley Professional, 1997.

作者简介

钱振江 男, 1982 年生于江苏苏州. 2013 年博士毕业于南京大学计算机科学与技术系, 讲师. 主要研究方向为操作系统安全、形式化验证和嵌入式系统.
E-mail: tony _ h@ sina. com

刘永俊 男, 1981 年生于江苏苏州. 现为常熟理工学院讲师. 主要研究方向为信息安全和嵌入式系统.
E-mail: yongjun1981@126. com

姚宇峰 男, 1981 年生于江苏苏州. 现为常熟理工学院讲师. 主要研究方向为信息安全和嵌入式系统.
E-mail: fengsingal81@ hotmail. com

汤 力 男, 1980 年生于江苏苏州. 现为常熟理工学院副教授. 主要研究方向为信息安全和嵌入式系统.
E-mail: kingtangli@ yahoo. com. cn

黄 皓 男, 1957 年生于江苏南京. 现为南京大学教授、博士生导师. 主要研究方向为系统软件、信息安全.
E-mail: hhuang@ nju. edu. cn

宋方敏 男, 1961 年生于江苏南京. 现为南京大学教授、博士生导师. 主要研究方向为数理逻辑和量子计算机.
E-mail: fmsong@ nju. edu. cn