软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(5):1204−1220 [doi: 10.13328/j.cnki.jos.005218]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Event-B 的航天器内存管理系统形式化验证*

乔 磊[1], 杨孟飞[2], 谭彦亮[1], 蒲戈光[3], 杨 桦[1]

[1](北京控制工程研究所,北京 100190)

[2](中国空间技术研究院,北京 100094)

[3](华东师范大学 计算机科学与软件工程学院,上海 200062)

通讯作者: 乔磊, E-mail: fly2moon@aliyun.com

摘 要: 内存管理系统位于操作系统内核的最底层,为上层提供内存分配和回收机制.在航天器这类安全攸关的关键系统中,其可靠性和安全性至关重要,必须要考虑到强实时性、有限空间限制、高分配效率以及各种边界条件约束.因此,系统通常采用较为复杂的数据结构和算法来管理内存空间,同时需要采用非常严格的形式化方法来保证航天器这类安全攸关系统的高可信性.对复杂内存管理系统的形式化验证也会比之前的验证工作带来更多难题,主要体现在:内存管理模块中的复杂数据结构的形式化描述,操作的规范语义,行为的建模,内部函数的规范及断言定义与循环不变式的定义,实时性验证等方面.针对这些问题,深入分析实际的航天器操作系统内存管理系统的特性;探索基于分层迭代的语义描述与验证的一般性方法与理论,并应用这些理论方法来验证一个具有实际应用的航天嵌入式操作系统的内存管理系统.该研究成果有望直接应用于我国新一代的航天器系统.

关键词: 航天器操作系统;内存管理;形式化验证

中图法分类号: TP311

中文引用格式: 乔磊,杨孟飞,谭彦亮,蒲戈光,杨桦.基于 Event-B 的航天器内存管理系统形式化验证.软件学报,2017,28(5): 1204−1220. http://www.jos.org.cn/1000-9825/5218.htm

英文引用格式: Qiao L, Yang MF, Tan YL, Pu GG, Yang H. Formal verification of memory management system in spacecraft using Event-B. Ruan Jian Xue Bao/Journal of Software, 2017,28(5):1204−1220 (in Chinese). http://www.jos.org.cn/1000-9825/5218.htm

Formal Verification of Memory Management System in Spacecraft Using Event-B

QIAO Lei[1], YANG Meng-Fei[2], TAN Yan-Liang[1], PU Ge-Guang[3], YANG Hua[1]

[1](Beijing Institute of Control Engineering, Beijing 100190, China)

[2](China Academy of Space Technology, Beijing 100094, China)

[3](School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China)

Abstract: In the safety-critical system of spacecraft, memory management system is the essential part of operating system kernels to provide allocation and cleanup mechanism at the lowest level and is obviously critical to the reliability and safety of spacecraft computer systems. The memory management system should satisfy strict constraints such as real-time response, space usage limit, allocation efficiency and many boundary conditions. It has to use very complex data structures and algorithm to manage the whole memory space. In order to make the complex memory management system of spacecraft highly reliable, the formal verification of the system also becomes much more complicated as it embodies formal verification of complex data structures such as two level segregated double linked list with two level bitmaps, specification of operations, modeling on behavior, assertion definition of inner function, loop invariant definition and real-time verification. This paper provides an in-depth analysis on these problems and characteristics of spacecraft memory management

* 基金项目: 国家自然科学基金(61502031, 61632005)
Foundation item: National Natural Science Foundation of China (61502031, 61632005)
收稿时间: 2016-08-30; 修改时间: 2016-09-25; 采用时间: 2016-12-07; jos 在线出版时间: 2017-01-20
CNKI 网络优先出版: 2017-01-20 16:06:42, http://www.cnki.net/kcms/detail/11.2560.TP.20170120.1606.020.html

© 中国科学院软件研究所 http://www.jos.org.cn

system to find certain general method and theory based on hierarchical iteration for verifying a concrete operating system used on spacecraft. The results of this study offer potential benefits in improving the reliability of the spacecrafts of China.

Key words: spacecraft operating system; memory management; formal verification

如今,操作系统已经广泛应用于航天器的控制系统中,其内核的安全可靠是构建高可信空间计算机系统的关键,因为任何一个微小的内核错误都有可能导致整个航天器系统的崩溃.此外伴随着系统软件内核的并行化、复杂化与追求高性能的发展趋势,内核的可靠性要求显得更加重要.然而现实情况却不容乐观,全球范围内的航天器会不断因为发现和公布的内核漏洞或设计缺陷而面临严重的威胁.形式化程序验证(formal program verification)[1−3]是使用逻辑推理系统来保证计算机程序正确性的方法之一.我们可以通过形式化的程序逻辑(program logic)来验证一个程序是否满足已给定的规范,并通过显式的证明使用户确信验证结果的正确性.对于规模相对较小而设计复杂的内核程序而言,形式化程序验证是保证其安全可靠的一种有效方法.

尤其是作为航天器操作系统内核的重要组成部分,内存管理系统的行为更是需要严格、精确、形式化地进行定义和验证.内存管理系统模块几乎为所有其他内核部分提供内存分配回收功能,位于整个操作系统内核的最底层(如图 1 所示).一旦这一部分代码存在任何纰漏或者缺陷,其引发的错误都有可能被放大到整个操作系统,危及航天器的运转,甚至导致无法挽回的安全事故发生.例如在 2004 年,美国航天局(NASA)耗资上亿美元的勇气号火星车突然连续多天与地面失去响应[4].通过 NASA 后续深入调查后得知,问题源于操作系统的存储系统模块,其内存动态分配策略存在着一个非常小的边界错误.该错误是由一个成熟的商业代码库引入的,并且成功躲过了火星车系统的密集测试与审查.然而当火星车在火星着陆后,这个错误在某个极端情况下触发,进而引发了一连串的系统错误,最终导致整个火星车系统瘫痪.

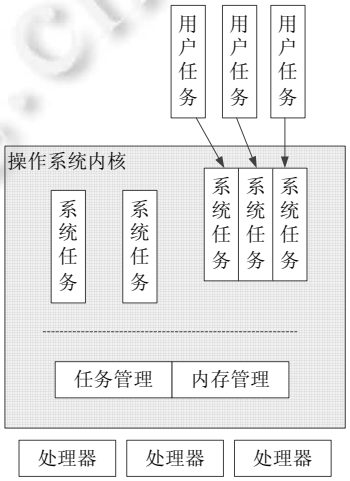


Fig.1 Structure of operating system

图 1 操作系统结构

1 相关验证研究

国外在形式化建模与验证方面开展了很多工作.比较著名的是法国计算机科学家 Leroy 领导的 CompCert 项目[5]验证了一个 C 编译器,对于编译程序的每一步操作都在辅助定理证明器 Coq 下给出了一个数学证明,确保代码语义在每一步都是不变的,从而证明 CompCert 编译器在整个编译过程中保持了代码的语义,确保源代码和目标代码的行为一致;德国教育研究部资助的 VeriSoft 及其后续 VeriSoft XT 项目[6]尝试验证包括操作系统的内核在内的从硬件到应用程序的完整的软件栈.由于涉及到商业秘密,很多内容没有公布,但是可以知道的是,他们证实了功能性验证软件栈的目标是可以实现的,并且在辅助定理证明器 Isabelle/HOL 中形式化验证了汇编一级的代码.中科大-耶鲁高可信软件联合研究中心研究人员在 Coq 中开发了一种 C 语言子集来形式化描述程序,付明等人在 uC/OS II 上做了很多研究工作,例如形式化验证其消息队列通信机制、任务调度等;梁红瑾等人提出了一种 RGSim 技术,用于支持并发程序间的精化关系的模块化验证.

操作系统内核中的内存管理模块从来都是一个逻辑复杂的代码,前面提到的形式化验证项目大都没有专门对实际使用的操作系统内存管理进行验证.耶鲁大学的 Yu 等人曾对一个 C 标准库中的 malloc/free 简化算法进行了严格的形式化验证[7].该算法由 72 行 MIPS 汇编语言编写,却用了超过 10 000 行的 Coq 代码,总共 239 个引理进行了形式化证明.在实际系统(如 Windows NT[8],OpenSolaris[9],Linux[10],FreeBSD[11])中部署的内存管理算法还要更加复杂,例如 Linux 内存采用的 Buddy 分配算法[12]、SLAB 分配算法[13−15]等.在实际系统中,内存管

© 中国科学院软件研究所 http://www.jos.org.cn

理算法的复杂性主要来源于以下几个方面.

(1) 内存分配需求的多样性.内核需要具有不同粒度、不同特性(例如是否共享、是否缓存)的内存块.

(2) 内存分配过程的性能考量.内存分配会以一个较高的频次被内核的其他模块调用.如何在内存池中较快地找到合适的内存块是内存分配算法的设计目标之一.一般来说,算法会采用较为复杂的数据结构来管理空闲内存池.

(3) 内存池中内存块碎片化管理.不同粒度的内存分配会导致内存的逐步碎片化,为了保证内存分配能够一直高效、稳定地运行,内存管理模块需要对一部分空闲内存块进行整理、分割、合并(如图 2 所示).

(4) 在航天器这类安全攸关的关键系统中,对内存管理算法还要考虑到实时性、空间限制、分配效率以及各种边界条件.通常分配算法要建立复杂的缓存区来保证内存管理过程的实时性需求.

内存管理系统模块的这些特征正是本文研究的重要出发点:我们需要采用非常严格的形式化方法来保证实际系统中负责内存管理的代码的可靠性安全性.对于航天器这类安全攸关系统而言,采用形式化方法来保证系统的可靠性安全性是必不可少的.

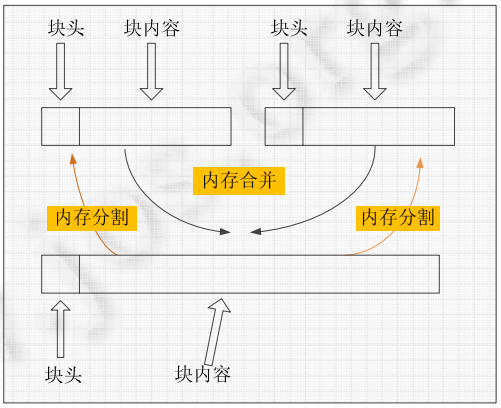


Fig.2 Split and merge operations in memory management

图 2 内存管理模块中的分割与合并

1.1 研究工作的现状与不足

如前所述,Yu 等人采用 Coq 语言作为形式化工具,验证了一种较为简单的内存分配算法的汇编代码实现.他们在Coq中定义了MIPS机器模型,针对汇编语言的类似Hoare逻辑的推理系统,内存分配算法的形式化规范,以及最终证明的性质.他们的工作为本文提供了一定的理论基础和验证经验,但是他们所验证的算法还远远不能满足航天器的要求.该内存分配算法过于简化,分配算法会导致过多的分割合并动作,实际应用性能较差,

在最近针对操作系统内核的形式化验证研究中,内存管理模块也被涉及到.首先是日本情报研究所对 Topsy 操作系统内存管理的验证[16].Topsy 内核是一个用于教学、侧重并发的大约 20K 行代码多线程微内核.Topsy 内核的内存管理模块采用堆管理的方式,提供基本的动态内存分配功能:内存初始化、内存分配、内存释放,并采用块链表来组织空闲内存池.他们也采用 Coq 作为形式化工具来定义该内存分配算法实现,采用分离逻辑来描述算法的断言与规范,最终在 Coq 中交互式地证明代码的正确性.该研究项目也为本文提供了一些理论基础,但是问题同样存在.Topsy 内核是一个用于教学的内核,其内存分配算法仍然过于简单,没有考虑现实系统中的诸多因素,其算法还不能在真正意义上成为实际有效的分配算法,同样不考虑实时性,不考虑分配效率问题.另一方面,其验证只是针对代码层面的验证,并未在更高层面进行验证,因此,验证方法落于具体实现细节,验证方法不具有普遍性.

另一个有名的操作系统内核验证工作是澳大利亚 NICTA 实验室对 seL4 微内核操作系统进行验证[17−19].他们创新性地采用一种函数式语言Haskell作为验证的中间工具,通过可运行的编程语言Haskell来定义算法的

© 中国科学院软件研究所 http://www.jos.org.cn

实现,由于函数式语言与定理证明语言的相似性,通过一个简单的转化,Haskell 语言编写的算法实现可以直接转变成内核 C 语言实现模块的形式化规范.但是,seL4 内核采用经典的微内核架构,将动态内存分配任务交由内核之外的用户级线程来实现.这样一来,seL4 就避开了内存分配算法的复杂性.而航天器目前采用的系统大多没有采用与 seL4 的微内核架构,也就是说,seL4 的研究方法并不适用于目前我国航天器上所采用的系统架构.另一方面,seL4 相对于 Topsy 进行了分阶段的验证,有了初步的分层概念,但是层次粒度较大,可靠性验证的说服力还有所欠缺.

1.2 研究概述与科学意义

SpaceOS[20]是国内第 1 个自主研发并经过空间飞行的空间飞行器嵌入式实时操作系统.SpaceOS 定位为空间飞行器专用的实时操作系统,具有系统资源占用低、实时性强以及功能精简的小、快、精的特点,已经应用在探月工程、新一代导航卫星、空间站等重大工程中.SpaceOS 虽然经过人工及自动化的测试,发现了一些明显的错误和不规范,但是否还存在一些潜在的漏洞和错误,仅靠测试的方法难以发现.为了验证 SpaceOS 操作系统的正确性,达到软件安全等级的最高级别,需要利用形式化的方法来验证.由于 SpaceOS 的设计模块性较好,可以分模块进行形式化验证.本文拟针对 SpaceOS 的内存管理模块进行验证.

以 VxWorks 为代表的国外空间操作系统大多采用基于单链表组织空闲块的首次适应算法进行内存分配,最坏情况的时间复杂度达到 $O(n)$量级.国内传统星载实时操作系统中的内存管理方法采用基于静态分区的内存管理方法,只能事先分配好固定大小的内存空间,而无法在实时创建动态任务时分配合适的内存空间,故灵活性不足且容易造成空间浪费.航天器嵌入式操作系统是强实时嵌入式操作系统,对内存算法的实时性要求很高.一般的动态内存分配算法存在着执行时间不确定与内存碎片过多等问题,在嵌入式实时系统中很少使用.TLSF (two level segregated fit)[21,22]动态内存算法具有内存分配和释放时间复杂度为常数,内存自动合并、灵活性强、内存碎片少等特点,已经在很多系统中得到应用,如 Amiga OS[23],Xtratum Hypervisors[24],Orocos[25]等. SpaceOS 操作系统也使用了该算法[20].本文将深入研究如何验证 TLSF 算法,从而保证 SpaceOS 操作系统对内存管理模块高效内存分配的可靠性要求.

TLSF 算法采用基于两级索引表结合分离表的动态内存管理方法,将时间复杂度从 $O(n)$降低到 $O(1)$量级,在时间平均性能以及确定性方面具有优越性;通过分离表的内存空间组织方式、基于二级位图的 good-fit 适配原则以及释放时立即回收结合的方法有效降低了内存碎片率.该方法不仅解决了动态任务创建问题,而且兼顾解决了动态内存管理灵活性与实时可靠性的矛盾.基于这种方法的空间飞行器实时多任务操作系统解决了共享计算资源下的实时调度、应用任务实时切换的问题,满足综合软件多任务并发、多个同优先级任务并存的任务调度要求,为复杂航天器多功能并行、顺利完成各项任务提供了基础保障.但同时,算法的复杂度也要比之前研究的内存管理算法增加不少,因而对其进行形式化验证也会比之前的工作复杂不少,带来验证难题,主要体现在:(1) 内存管理模块中的复杂数据结构的形式化描述;(2) 内存管理模块接口函数和操作的规范语义;(3) 内存管理代码的内部函数的规范与断言定义与循环不变式的定义;(4) 内存管理操作的实时性验证.

本文针对当前研究的不足,深入分析实际的操作系统内存管理的特性,探索研究基于分阶段分层次自上而下循环迭代的内存管理的语义描述与验证的一般性方法与理论,并应用这些理论方法,以验证一种具有实际应用的航天器嵌入式操作系统 SpaceOS 的内存管理算法以及代码实现.本文是对实际嵌入式操作系统系统可靠性验证的深入探索,具有较好的理论价值与科学意义,该方法将可以作为验证操作系统其他模块的普适性方法,研究成果有望直接应用于我国新一代的航天器系统,具有较高的实用价值.

2 基于 Event-B 方法的内存管理系统建模与验证方法

围绕上文中提出的关键问题,采取如下思路:验证过程基于 Rodin[26]建模工具利用 Event-B[26,27]数理抽象方法,对航天器嵌入式操作系统内存管理模块进行形式化建模.对所产生的模型,验证其是否能够满足航天嵌入式操作系统内存管理模块需求中所提炼出的性质,从而说明操作系统内存模型的正确性.在建模过程中,对内存管理需求,设计以及实现分层次自顶向下进行形式化建模.

© 中国科学院软件研究所 http://www.jos.org.cn

按照软件工程开发过程划分为需求分析、设计、编码实现这 3 个主要阶段.每个阶段都要根据已有的文档资料进行形式化描述,建立模型,分阶段进行验证,阶段内要保证正确性,阶段间要保证一致性.

在第 1 阶段中,根据需求说明文档进行功能和性质的提取,建立不同模块的模型.主要关注的是用户需求中提出的功能操作要满足设计需求描述的性质,以及功能和性质本身的正确性.第 2 阶段是将模型以分层的形式逐步精化得到设计模型,这样可以保证上下层之间的一致性.对设计模型继续精化,将具体数据结构等实现方式作为精化依据,可进一步得到实现模型.在第 3 阶段,从 C 代码实现提取一个抽象化的模型与实现模型进行一致性验证,可以判定 C 模型是否与实现模型一致.以实现模型作为正确实现的基础,使用 Hoare 逻辑对 C 代码验证,对发现不一致的问题提出改进,找到原有代码的漏洞并进行修正.

该验证过程基于 Rodin 建模工具,利用 Event-B 数理抽象方法,对航天嵌入式操作系统内存管理模块进行形式化建模.对所产生的模型,验证其能否满足航天嵌入式操作系统内存管理模块需求中所提炼出的性质,从而说明操作系统内存模型的正确性.在建模过程中,对内存管理需求、设计以及实现都分别进行形式化建模.

2.1 内存分配算法分析

TLSF 算法采用较好匹配(good-fit)原则.较好匹配原则是想尽量达到最佳匹配的效果(产生的碎片最少),选取最佳匹配的内存块往往比选取较好匹配的内存块花的步骤更多,因此选取较好块更合适.本项技术采用将分组空闲链表(segregated list)和位图匹配(bitmaps fit)两种机制结合的方法.位图的使用使执行速度较快.

2.1.1 数据结构

TLSF 算法的数据结构[28]可以用二维数组表示.第 1 维数组将空闲内存块按照 2 的 n 次幂的大小划分,例如 16,32,64,128 等;第 2 维将第 1 维的内存块进行线性划分,而划分的个数(定义为二级索引,SLI)是用户可配置的参数.2^{SLI} 不应该超过 32,以便一个 32 位的位图可以表示所有范围空闲块是否可用.一般地,SLI 为 4,最多为 5.在图 3 给出的数据结构中,SLI=2.

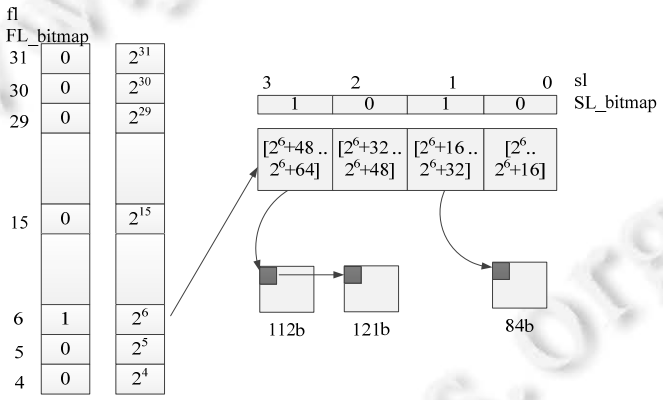


Fig.3 Data structure of free block

图 3 空闲数据块结构

图 3 显示了两级的数据结构.第 1 级是一组指针,指向二级空闲块链表.按照 FL_bitmap 数组可以知道第 1 级的哪些位有空闲块,比如图中 FL_bitmap[6]为 1,则表示在[2^6,2^7]这一级,这个大小范围有空闲块.二级位图 SL_bitmap 将一级块大小范围按 4 段划分,分别表示各段是否有空闲块.

2.1.2 重要参数

本项技术的结构特性主要依靠 3 个参数.

(1) 一级索引(FLI):用于标识一级链表长度,一级链表长度是按类分的,也就是类的个数.每个类按照 2 的 n 次幂增长.FLI 不是设计者一开始就可以设定好的,而是程序初始化时动态计算的,FLI 要根据空闲内存的大小计算得到,如下所示:

© 中国科学院软件研究所 http://www.jos.org.cn

$$FLI = \min(\log_2^{(memory_pool_size)}, 31).$$

(2) 二级索引(SLI):将一级内存块大小范围按照线性子划分.SLI 由设计者设定.例如,*SLI*=4,则将每一个一级链表按 16 段进行划分.SLI 越大,则划分得越细,但一般而言,SLI 不超过 5.因为 SLI 超过 5,则 2 的 SLI 次方超过 32,用位图方式就不能表示,不能使用按字位图处理指令.

(3) 最小块大小(MBS):这个参数是定义最小块的大小.所谓最小块,即不能再划分的块,一般为 16 位.

根据 FLI,SLI 和 MBS 的初始化,其他参数也可以确定,参数定义如下.

(1) 总的链表数目: $2^{SLI}(FLI - \log_2^{(MBS)})$.

(2) 链表连接的内存块的大小所属的范围.

$$\left[2^i + \left(\frac{2^i}{2^{SLI}} \cdot j\right), next_size\right],$$

其中,

$$\forall i, j \in N^+ \wedge (\log_2^{(MBS)} \leqslant i \leqslant FLI) \wedge (0 \leqslant j \leqslant 2^{SLI}),$$

并且

$$next_size = \begin{cases} 2^{i+1} - 1, & \text{if}(j = 2^{SLI}) \\ 2^i + \left(\dfrac{2^{i^{SLI}}}{2} \cdot (j+1)\right), & \text{otherwise} \end{cases}.$$

2.1.3 算法逻辑

大部分操作依赖 *segregation_list*()映射函数.给出块大小,用映射函数计算出参数 *f* 和 *s*,然后利用这两个参数找到请求块在两个链接表数组的相应位置.这个函数可以通过按位搜索指令(适用于大部分现代处理器),并且利用一些数学公式高效执行.

算法结构通过下列操作执行.

(1) 初始化结构:这个函数初始化算法的数据结构,初始化空闲内存.

它接受 3 个参数:一个指向内存池的指针,内存池的大小,二级索引.创造几个相互独立的内存池是可能的.

(2) 获取一个空闲块:返回一个内存块(请求大小或者大一些)的指针.通过请求大小找到对应的列表.操作步骤如下.

第 1 步.计算出 *f* 和 *s*,找到对应的列表,如果列表为不空,则将列表的头节点取下来,标识为 busy,然后将地址返回;否则,

第 2 步.在算法数据结构中,搜索下一个(比请求大小大)不为空的链表.利用位图标志完成这个搜索所需时间为常数量.如果找到一个链表不为空,则链表的头节点可以用来满足要求.由于这个块比请求大小要大,所以有必要拆分一下,并将剩余的内存块插入到相应的链表.另一方面,如果没有发现不为空的链表,则请求失败.

(3) 插入一个空闲块:这个函数在算法结构中插入一个空闲块.那个映射函数用来计算 *f* 和 *s*,然后根据这两个参数找到内存块的插入位置.

(4) 连接内存块:使用分界标签技术,可以判断前一个内存块是空闲还是忙.如果这个块是空闲状态,则从链表中移除这个块,并与现在的块合并.下一个物理块也做同样的操作.一旦与相邻的空闲块合并,就将这个新块插入到合适的链表中.

2.2 基于单内存块和内存池对象的内存管理行为模型

内存管理的基本对象我们定义为单个内存块和整个内存池,首先要确定上述两个基本对象的环境 context.

2.2.1 内存池对象模型

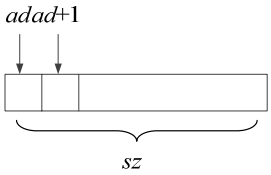
可描述为:内存池是由一些存储在地址中的数值构成的,其逻辑描述如下:

$ad \mapsto v$:*ad* 表示一个地址,*v* 表示一个数值.

© 中国科学院软件研究所 http://www.jos.org.cn

2.2.2 内存块对象模型

可描述为(ad 为起始地址,sz 为大小)



$$\textbf{Block}\ ad\ sz := (sz = 0 \wedge \text{emp}) \vee (sz > 0 \wedge (\exists e \cdot (ad \mapsto e)) \times (\textbf{Block}\ (ad+1)\ (sz-1))).$$

2.2.3 行为及属性

对于内存块对象只有两种操作行为:分配、回收.另外需要一个状态属性,用以标志其在经过不同操作行为后的状态,以确定各种性质是否得到满足.定义一个内存块在被分配之后其状态就为 busy 状态,否则回收后为 free 状态.其次,还要有空间大小属性,用以描述内存块的尺寸.所有空闲状态的内存块在 TLSF 算法中以双向链表的形式组织起来,这种方式非常方便处理移除内存块.如图 4 所示.

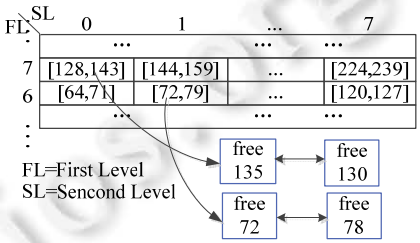


Fig.4 Form of the doubly linked list for free block

图4 空闲内存块的双向链表组织形式

对于图 4 中的一个空闲内存块案例,其形式化描述如下:

$$\textbf{Freeblocklist}\ x := \exists st \cdot (x \mapsto st, nil) \vee \exists next \cdot (next \neq nil) \wedge$$
$$(x \mapsto free, next) \times (\textbf{Block}\ (x+2)\ (next - x - 2)) \times (\textbf{Freeblocklist}\ next).$$

有了内存块对象描述及属性描述,整个内存管理的行为模型可以构建出来,如图 5 所示.

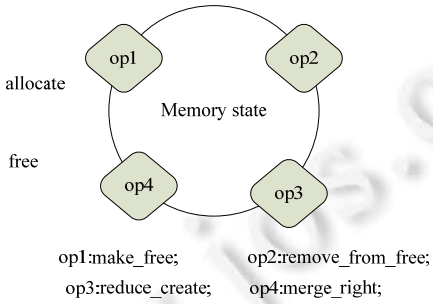


Fig.5 Behavioral model of memory management

图 5 内存管理的行为模型

2.3 基于内存对象空间大小及其状态属性组合的行为及性质描述

如第 2.2 节所述,将内存对象定义为单个内存块及整个内存池,内存管理的所有行为将基于这两个对象开展.内存对象要具有空间大小和状态属性,标志基于其上的操作行为是否可以进行.在内存管理行为针对内存对象操作的行为模型基础上,至少要保证如下的一些性质.

© 中国科学院软件研究所 http://www.jos.org.cn

(1) 状态为占用的内存块不能被再分配.

(2) 状态为空闲的内存块不能被释放.

(3) 两块状态为占用的内存块不能重叠.

(4) 任意两块毗邻的内存块不能重叠.

(5) 两块空闲块之间至少有 1 块非空闲块.

(6) 若内存块状态改变则立即更新内存管理中的空闲块数据结构.

(7) 提供用户分配内存和释放内存接口.

(8) 分配内存块时能够选择最合适的内存块.

(9) 分配的内存块若大于请求的内存块,则需要对分配的内存分割.

(10) 当分配不成功时返回错误信息.

(11) 判断传入调用释放接口的内存块是否合法(必须为非空闲内存块).

(12) 提供对指定内存区的动态管理,可根据要求分配指定大小的内存,当内存使用完成之后,内存管理模块可提供对内存的释放功能.

(13) 如果内存区中存在大小为 N 个字节的连续地址空间,对任意一个申请内存大小小于等于 N 个字节的分配请求都是可被满足的.

(14) 对于任意内存分配请求,如果请求被满足,则被分配的内存空间的地址是连续的.

(15) 任意已被分配的内存空间都是可被释放的.

(16) 任意已被释放的内存空间都是可被分配的.

(17) 内存的最小碎片的大小是大于等于某固定值.

(18) 内存的碎片率小于或等于某固定值.

(19) 任意大小内存空间的分配请求的执行时间小于等于某一固定值 M.

(20) 任意已分配内存空间的释放操作执行时间小于等于某一固定值 N.

(21) 在任意次数内存分配、释放操作之后,执行一次内存分配或内存释放操作的执行时间仍小于等于固定值 M 或 N.

但上述性质是否完备还需要开展深入的工作,寻找理论上的创新和突破.

2.4 基于分层模型精化和迭代的验证方法

有了上述行为模型,即可开展具体的验证工作,证明内存管理的各种行为是满足安全可靠性质的.针对航天器操作系统的软件工程化设计及开发特点,依据内存管理需求描述和设计思路,建立内存管理模块的分层次形式化模型,并通过精化,最终生成可执行的代码模型,并证明生成代码与已有代码间的等价性.

本验证过程基于 Rodin 建模工具利用 Event-B 数理抽象方法,对航天器嵌入式操作系统内存管理模块进行形式化建模.在 Rodin 工具中所建的模型都由两部分组成:context(环境)和 machine(机器).一个环境可能包含载体集合(carrier set)、常量(constant)、公理(axiom)和定理(theorem).载体集合是用户定义类型;常量表示目标系统中的不变量;公理用于假设载体集合以及常量的性质;定理是从载体集合以及常量推导出的性质.载体集合和常量是模型的参数.机器用于描述 Event-B 的行为.机器与环境相连后能够看到相应环境中的信息.机器 M 看到(sees)环境 C 意味着它可以访问环境 C 中的载体集合 s 以及常量 c,例如,在模型中对其进行引用,或者在证明中将环境 C 中的公理 $A(s,c)$和定理 $T(s,c)$作为假设条件.机器 M 可能包含变量(variable)、不变式(invariant)、定理、变体(variant)以及事件(event).变量 v 定义了一个机器的状态,其变量约束用不变式 $I(s,c,v)$表示.定理作为变量 v 所需满足的额外性质.Event-B 机器的一个最重要特性是它包含不变式 I,不变式是机器所有可达状态都必须满足的性质.显然,对任何机器,其不变式是无法先验满足的,因而必须要对其做验证.

在本文中,模块的描述将操作系统内存管理需求对模型的不同精化层次进行划分描述.需求模型和设计模型没有完全的界限,根据精化步骤,拟将模型划分为 10 层,每层覆盖不同的需求和验证不同的性质.从需求到设计所建的形式化模型主要如表 1 所示.

© 中国科学院软件研究所 http://www.jos.org.cn

Table 1 Formalized model

表1 形式化模型

序号	模型名字	模型说明
1	模型 1(C0,M0)	本模型将实际物理内存池形式化描述为无限离散的集合,将内存块抽象描述为无限离散的集合,将内存块与实际所对应的内存池描述为内存块到内存池的映射关系.每个块映射到内存池的不同部分,且每个块的映射值域交集为空来表示每个块不会重叠.
2	模型 2(C1,M1)	每个内存块逻辑上都是有起始位置和结束位置的,实际情况下是起始地址和末尾地址,两者之间的间距认为是块的大小.因此,本模型在模型 1 的基础上进行精化,将整个内存池的抽象为一个连续的有具体长度的一段自然数.
3	模型 3(C2,M2)	实际情况下,内存块的数目是有限的,在模型 2 的基础上,精化内存块集合的数目为有限,验证每个块是不会重叠的安全性质.
4	模型 4(C3,M3)	在实际的内存管理设计中,在进行分配和释放操作过程中,需要先计算每个块在一个索引表中的位置,本模型将每个块到索引表中的位置精化表示为函数映射.
5	模型 5(C4,M4)	在实际的内存管理设计中,在分配的情况下,需要判断分配请求的大小范围之后再进行分配,否则会存在安全性问题.本模型加入了分配大小范围判断.
6	模型 6(C4,M5)	在分配情况下,总共有两种可能:分配成功和分配不成功;在释放的情况下,总共有两种可能:释放成功和释放不成功.本模型在上一层模型的基础上进行精化,避免分配和释放死锁的情况发生.
7	模型 7(C4,M6)	本模型是链表精化初步模型,实际情况下为不同等级和类别的空闲块为一个双向链表.精化加入集合表示链表所有元素.本模型也刻画了链表的加入和删除操作.
8	模型 8(C5,M7)	由原有离散的集合精化加入链表中首尾元素,并且加入单向链表的表示.
9	模型 9(C5,M8)	在原有的模型基础上,加入原来单向链表的逆向链表.
10	模型 10(C5,M9)	在此模型中,因为集合已经用链表表示,本模型删去了链表元素组成的集合.

从需求到设计的形式化模型可按图 6 所示的从左到右的过程逐步进行构建.模型 M0-M5 为需求模型,其中 M0 覆盖用户需求,模型 M1~模型 M5 覆盖设计需求.模型 M6~模型 M9 为设计模型,描述了具体的数据结构和实现方式.生成的 code.c 为底层模型转换而来,original.c 为目标代码.

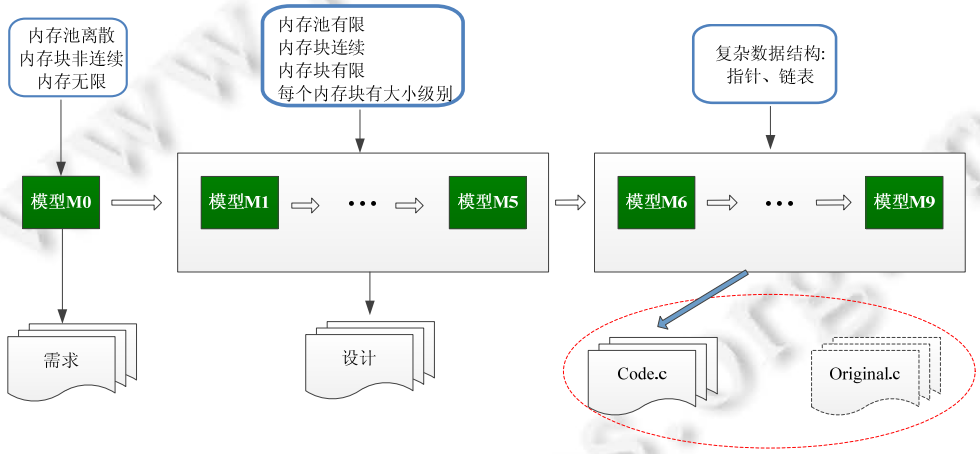


Fig.6 Modular refinement flow path

图 6 模块精化技术流程

3 内存管理模型建模及精化验证

下面以 M0~M6 层为例介绍模型逐层精化的描述与验证过程,后面几层模型精化验证工作还在进行中.

3.1 模型M0

本模型将实际物理内存池形式化描述为无限离散的集合,将内存块抽象描述为无限离散的集合,将内存块与实际对应的内存池描述为内存块到内存池的映射关系.每个块映射到内存池的不同部分,且每个块的映射值域交集为空以表示每个块不会重叠.

© 中国科学院软件研究所 http://www.jos.org.cn

3.1.1 模型 M0 说明

把整个内存抽象为自然数的一个子集 M,记作 $M \subseteq \mathrm{NAT}$, 并且改内存集合 M 是有限的集合(用函数 $\mathrm{finite}(M)$ 表示有限).因为内存是实际存在并且是真实具有的,所以该 M 集合还必须加上属性 $M \neq \varnothing$.

内存分配是以块为单位的,一个块占据了整个内存块的一部分.可以把内存块的指向变量抽象为自然数的一个子集,把块占据实际内存一部分表示为 B 到 M 的映射关系.把能够映射到 M 集合的标记量归为集合 $block$(满足 $block \subseteq B$).在具体情况下中,整个 M 可能就是一个整块,所以 $block$ 到 M 的映射关系为

$$mem \in block \rightarrow P(M).$$

函数 mem 就是表示 $block$ 集合到 M 幂级的映射,带入类型为 $block$ 的参数,返回的是 M 的一个幂级.因为所有 $block$ 所指向的整个值域为整个 M 集合,即 mem 的值域为 M.所以加入不变式:

$$M = union(ran(mem)).$$

ran 函数表示一个映射的值域,$union$ 为集合的并集.因为 $block$ 的类型为 $busy$ 或者 $free$,表示一个块状态为被占用或者空闲,用 $free$ 表示空闲块集合, $free \subseteq block$. 两个不相等的块都不能有重叠(体现了性质 3 和性质 4),因此需要加入以下不变式:

$$\forall b1, b2 \cdot b1 \in block \wedge b2 \in block \wedge b1 \neq b2 \Rightarrow mem(b1) \cap mem(b2) = \varnothing.$$

3.1.2 模型 M0 初始状态

模型的初始状态是所有的 M 就是一个块,即 B 中的一个元素 b 映射到 M 整个集合.$block$ 和 $free$ 集合都只有 1 个元素,初始的状态为以下状态.

$$block := \{b\},\ free := \{b\},\ mem := \{b \mapsto M\}.$$

为了描述性质 7,模型有两个基本操作:分配和释放.在本层模型中使用离散数学的集合元素关系来表示内存的分配和释放操作.只以分配操作为例来说明.

分配操作是从空闲内存块中分离出一个小于其大小的内存块使用.一般情况下,需要从一个空闲内存块 bl 中分离出一个大小为 s 的内存块 cl,可以表示为:

$$block := block \cup \{cl\},$$

$$free := (free \cup \{cl\}) \setminus \{bl\},$$

$$mem := (\{bl\} mem) \cup \{bl \mapsto s, cl \mapsto mem(bl) \setminus s\}.$$

在 Rodin 平台下易证,分配操作后的变量状态仍然使前面的不变式成立.M0 层证明义务统计如图 7 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m0	**36**	**31**	**5**	**0**	**0**
INITIALISA...	6	6	0	0	0
inv1	4	4	0	0	0
inv2	4	3	1	0	0
inv3	4	1	3	0	0
inv4	6	6	0	0	0
inv5	5	5	0	0	0
allocate1	9	6	3	0	0
allocate2	2	2	0	0	0
inv6	5	4	1	0	0
free1	1	1	0	0	0
thm	2	2	0	0	0
free2	7	6	1	0	0
free3	7	6	1	0	0

Fig.7 Statistics of proof obligation for M0

图 7 M0 层证明义务统计

3.2 模型M1

3.2.1 模型 M1 说明

每个内存块逻辑上都有起始位置和结束位置,实际情况下是起始地址和末尾地址,两者之间的间距认为是块的大小.因此本模型在模型 1 的基础上进行精化,将整个内存池抽象为一个连续的有具体长度的一段自然数.

© 中国科学院软件研究所 http://www.jos.org.cn

因为内存块都逻辑上都是有起始位置和结束位置的,实际情况是起始地址和末尾地址,两者之间的间距被认为是块的大小.因此,将初始模型中表示整个内存的集合 M 定义为一段连续的有具体长度的自然数.

$M = 1..m(m \in \mathbb{N}1)$ 表示 M 是 1~m 的区间,表示实际大小为 m 的内存块.因为每个块的标示量都有一个起止地址.定义不变式:

$first \in block \rightarrow \mathbb{N}$, //起始地址是块到自然数的映射

$last \in block \rightarrow \mathbb{N}$, //末尾地址是块到自然数的映射

$\forall bl \cdot bl \in block \Rightarrow mem(bl) = first(bl)..last(bl)$. //每个块的映射值域为块的起始地址到末尾地址

定义以下定理,这些定理可以由以上不变式推导出来:

$\forall bl \cdot bl \in block \Rightarrow first(bl) \leqslant last(bl)$, //块的地址排列是顺序的

$first \in block \rightarrowtail \mathbb{N}$, //起始地址只有 1 个

$last \in block \rightarrowtail \mathbb{N}$, //末尾地址只有 1 个

$\mathrm{ran}(first) \subseteq M$, //起始地址也属于块的一部分

$\mathrm{ran}(last) \subseteq M$, //末尾地址也属于块的一部分

$\forall bl \cdot bl \in block \wedge first(bl) \neq 1 \Rightarrow first(bl) - 1 \in ran(last)$, //块与块是连续的,一个块的起始地址的前一个为一个块的末尾地址

$\forall bl \cdot bl \in block \wedge last(bl) \neq m \Rightarrow last(bl) + 1 \in ran(first)$, //块与块是连续的,一个块的末尾地址的后一个为一个块的起始地址.

3.2.2 初始模型状态

模型的初始状态是所有的 M 就是一个块,地址范围是 1~m,即 B 中的一个元素 b 映射到 M 整个地址区间.在初始状态下加入:

$$first := \{b \mapsto 1\},$$
$$last := \{b \mapsto m\}.$$

由于在分配和释放事件中,M0 层的 *mem* 与 M1 层的 *first* 和 *last* 存在等价替换关系:$men(b)=last(b)-first(b)+1$.因此可以使用 *first* 和 *last* 的映射关系替代 *mem*.M1 层证明义务统计如图 8 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m1	**62**	**50**	**12**	**0**	**0**
INITIALISA...	3	3	0	0	0
allocate1	12	9	3	0	0
inv1	7	7	0	0	0
inv2	7	6	1	0	0
inv3	8	4	4	0	0
allocate2	3	3	0	0	0
thm1	2	2	0	0	0
free1	0	0	0	0	0
thm2	1	0	1	0	0
thm3	1	0	1	0	0
thm4	1	0	1	0	0
thm5	1	0	1	0	0
thm6	2	1	1	0	0
thm7	2	1	1	0	0
free2_1	6	5	1	0	0
free2_2	6	5	1	0	0
free2_3	6	6	0	0	0
free2_4	6	5	1	0	0
free3	9	9	0	0	0

Fig.8 Statistics of proof obligation for M1

图 8 M1 层证明义务统计

3.3 模型M2

3.3.1 模型 M2 说明

为了验证性质:内存中没有毗邻的空闲内存块(性质 5 的另一种表示),需要定义一个布尔型变量 *merged*, *merged*=True.没有相邻的空闲块意味着空闲块的先后块不是空闲的,即有以下两个不变式:

© 中国科学院软件研究所 http://www.jos.org.cn

$$merged = \text{TRUE} \Rightarrow (\forall bl \cdot bl \in block \wedge first(bl) > 1 \wedge bl \in free \Rightarrow last^{\sim}(first(bl) - 1) \notin free),$$

$$merged = \text{TRUE} \Rightarrow (\forall bl \cdot bl \in block \wedge last(bl) < m \wedge bl \in free \Rightarrow first^{\sim}(last(bl) + 1) \notin free).$$

在内存释放后,如果分配后的内存与前后两个内存块出现的都是空闲块且地址相邻,就应当置 *merged*=FALSE,触发内存块合并事件,将相邻空闲内存块合并后再置 *merged*=True.

当 *merged* 为 TRUE 时,内存中就不会有毗邻的空闲内存块.

3.3.2 初始模型状态

初始状态下,只有 1 个空闲内存块,没有毗邻空闲内存块.在初始状态下加入:

$$merged = \text{True}.$$

在本层模型中通过引入 *merged* 变量增加了释放内存块后是否存在两个毗邻内存块的情况,如果存在,则执行新操作,将两个空闲内存块合并成一个.M2 层证明义务统计如图 9 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m2	**35**	**21**	**14**	**0**	**0**
INITIALISA...	3	3	0	0	0
allocate1	2	0	2	0	0
allocate2	2	2	0	0	0
free1_1	3	1	2	0	0
inv1	13	6	7	0	0
inv2	13	6	7	0	0
free1_2	3	2	1	0	0
free1_3	3	2	1	0	0
free1_4	3	3	0	0	0
free2_1	6	5	1	0	0
free2_2	2	0	2	0	0
free2_3	2	1	1	0	0
free2_4	2	0	2	0	0
free3	2	0	2	0	0
inv3	1	1	0	0	0

Fig.9 Statistics of proof obligation for M2

图 9 M2 层证明义务统计

3.4 模型M3

3.4.1 模型 M3 说明

实际的数据结构如图 4 所示,每个块都有一个大小范围,属于一个类别里.将这个大小级别抽象,如图 10 所示.

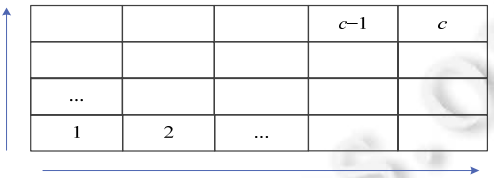


Fig.10 Abstract 2D array

图 10 抽象二维数组

因为每个块都能计算块所处于的大小范围,我们定义映射函数:

$$g \in 1..m \rightarrow\!\!\!\!\rightarrow 1..c, \text{其中}, c \in \mathbb{N}1.$$

函数 g 用来计算块的大小级别,传入参数是块的大小.由于抽象二维数组每个元素都连着一个大小级别的空闲块的双向链表.我们定义: $box \in free \rightarrow 1..c$, 表示空闲块到二维数组中位置的映射.因此,有空闲块按其大小分配在某一个大小级别中: $\forall fb \cdot fb \in free \Rightarrow box(fb) = g(last(fb) - first(fb) + 1)$.

3.4.2 初始模型状态

初始状态下,整个内存块映射到最大的块级别.在初始状态下加入: $box := \{b \mapsto c\}$.

在分配和释放事件中,我们需要更新二维数据结构的结构信息.在分配一个块的一部分时,需要我们更新

© 中国科学院软件研究所 http://www.jos.org.cn

box 的映射关系.如我们分配一个块 bl 的一部分时,余下的为 cl,q 为分配的大小.分配之后 bl 不再位于 $free$ 集合中,所以从 box 的值域中移除,cl 为新的空闲块,所以加入到 box 的值域,并且映射到相应的大小级别: $box := (\{bl\} <+ box) \cup \{cl \mapsto g(last(bl) - first(bl) + 1 - q)\}$.

M3 层证明义务统计如图 11 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m3	**37**	**13**	**24**	**0**	**0**
INITIALISA...	2	2	0	0	0
allocate1	4	0	4	0	0
allocate2	3	3	0	0	0
free1_1	3	2	1	0	0
free1_2	3	2	1	0	0
free1_3	3	1	2	0	0
free1_4	3	3	0	0	0
free2_1	3	0	3	0	0
free2_2	3	0	3	0	0
free2_3	3	0	3	0	0
free2_4	3	0	3	0	0
free3	3	0	3	0	0
inv1	12	6	6	0	0
inv3	13	3	10	0	0

Fig.11 Statistics of proof obligation for M3

图 11 M3 层证明义务统计

3.5 模型M4

为了验证每个块是不会重叠的安全性质,需要描述一个内存块的前后块,定义:

$preb \in block \to block$, //表示一个块的相邻前块

$next \in block \to block$. //表示一个块的相邻后块

3.5.1 模型 M4 说明

两个相邻块不会重叠意味着相邻块的地址是连续的.

$$\forall bl \cdot bl \in block \wedge (first(bl) \neq m) \Rightarrow last(preb(bl)) = first(bl) + 1,$$

$$\forall bl \cdot bl \in block \wedge (last(bl) \neq m) \Rightarrow first(nexb(bl)) = last(bl) + 1.$$

这样,在 M4 中通过不变式定义的内存模型就是一个连续的相邻块无重叠的地址空间.

3.5.2 初始模型状态

初始状态下,整个内存池只有一个内存块,其前后相邻内存块都为空.在初始状态下加入:

$$preb := \{b \to \text{null}\},$$

$$next := \{b \to \text{null}\}.$$

分配和释放事件中,需要更新内存块的前后块映射关系.

M4 层证明义务统计如图 12 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m4	**16**	**16**	**0**	**0**	**0**
INITIALISA...	1	1	0	0	0
allocate1	4	4	0	0	0
allocate2	1	1	0	0	0
free1_1	1	1	0	0	0
free1_2	1	1	0	0	0
free1_3	1	1	0	0	0
free1_4	1	1	0	0	0
free2_1	1	1	0	0	0
free2_2	1	1	0	0	0
free2_3	1	1	0	0	0
free2_4	1	1	0	0	0
free3	1	1	0	0	0
inv1	13	13	0	0	0

Fig.12 Statistics of proof obligation for M4

图 12 M4 层证明义务统计

© 中国科学院软件研究所 http://www.jos.org.cn

3.6 模型M5

在模型 M3 中只描述了块到块大小的映射,为了描述相同大小的块需要定义一个新映射关系 $cs \in 1..c \rightarrow \mathbb{P}(block)$.

3.6.1 模型 M5 说明

满足上面映射关系的块是空闲的.

$$\forall x, bl \cdot (x \in 1..c) \wedge (bl \in cs(x)) \Rightarrow (box(bl) = x) \wedge (bl \in free),$$

$$\forall bl \cdot bl \in ran(cs) \Rightarrow bl \cap (block \setminus free) = \varnothing.$$

不同大小的块不会指向相同的块号.

$$\forall x_1, x_2 \cdot (x_1 \in 1..c) \wedge (x_2 \in 1..c) \Rightarrow cs(x_1) \cap cs(x_2) = \varnothing.$$

3.6.2 初始模型状态

初始状态下,整个内存池只有 1 个空闲内存块.在初始状态下加入: $cs := \{c \rightarrow \{b\}\}$.

在分配和释放事件中,需要更新内存块的前后块映射关系.M5 层证明义务统计如图 13 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m5	**4**	**4**	**0**	**0**	**0**
INITIALISA...	0	0	0	0	0
allocate1	0	0	0	0	0
allocate2	0	0	0	0	0
allocate3	1	1	0	0	0
free1_1	0	0	0	0	0
free1_2	0	0	0	0	0
free1_3	0	0	0	0	0
free1_4	0	0	0	0	0
free2_1	0	0	0	0	0
free2_2	0	0	0	0	0
free2_3	0	0	0	0	0
free2_4	0	0	0	0	0
free3	0	0	0	0	0
free4	0	0	0	0	0
DLF1	2	2	0	0	0
DLF2	1	1	0	0	0

Fig.13 Statistics of proof obligation for M5

图 13 M5 层证明义务统计

3.7 模型M6

模型 M6 是链表精化初步模型,实际情况下,不同等级和类别的空闲块为一个双向链表.精化加入集合表示链表所有元素,同时刻画了链表的加入和删除操作.

定义空闲内存块的前后块.

$pref \in free \rightarrow free$, //表示一个空闲块的相邻前空闲块

$nexf \in free \rightarrow free$. //表示一个空闲块的相邻后空闲块

为了描述 TLSF 分配算法中查找合适大小的空闲内存块,需要定义内存块大小到相应空闲块的映射关系.

$head \in 1..c \rightarrow free$, //表示内存块大小到第 1 个空闲块的映射

$tail \in 1..c \rightarrow free$. //表示内存块大小到最后一个空闲块的映射

3.7.1 模型 M6 说明

TLSF 算法的二维链表中每一个内存大小都对应一个一维链表,该链表可能有多个空闲块.

$$\forall bl \cdot bl \in cs(box(bl)) \wedge card(cs(box(bl))) > 1 \Rightarrow (pref(bl) \in cs(box(bl)) \wedge pref(bl) \neq bl) \wedge$$

$$(nexf(bl) \in cs(box(bl)) \wedge nexf(bl) \neq bl).$$ //即链表中的内存块前后块都不是它本身

对于某一大小的内存块链表中两个不同内存块,它们的前后块各不相同.

© 中国科学院软件研究所 http://www.jos.org.cn

$$\forall b_1,b_2 \cdot b_1 \in cs(box(b_1)) \wedge b_2 \in cs(box(b_1)) \wedge card(cs(box(b_1))) > 1 \Rightarrow$$
$$(pref(b_1) \neq pref(b_2)) \wedge (nexf(b_1) \neq nexf(b_2)),$$ //表示内存块映射是唯一的

该内存块链表的头不存在前一个相邻空闲块,链表尾也不存在后一个相邻空闲块.

$$\forall x \cdot x \in 1..c \wedge card(cs(box(x))) > 1 \Rightarrow (pref(head(x)) = \text{null}) \wedge (nexf(tail(x)) = \text{null}).$$

该链表可能只有 1 个空闲块.

$$\forall bl \cdot bl \in cs(box(bl)) \wedge card(cs(box(bl))) = 1 \Rightarrow (pref(bl) = \text{null}) \wedge (nexf(bl) = \text{null}),$$

//即链表中的内存块前后块都为空

3.7.2 初始模型状态

初始状态下,整个内存池只有 1 个空闲内存块,其前后相邻空闲块都为空.在初始状态下加入:

$$pref := \{b \mapsto \text{null}\},$$
$$nexf := \{b \mapsto \text{null}\},$$
$$head := \{c \mapsto b\},$$
$$tail := \{c \mapsto b\}.$$

在分配和释放事件中,需要更新二维链表中空闲内存块的前后块映射关系以及链表头尾内存块的映射关系.

M6 层证明义务统计如图 14 所示.

Rodin Problems | Properties | Tasks | Statistics

Element Name	Total	Auto	Manual	Reviewed	Undischarged
M2m6	**2...**	**1...**	**50**	**0**	**0**
INITIALISA...	15	14	1	0	0
allocate1	16	11	5	0	0
allocate2	13	11	2	0	0
allocate3	0	0	0	0	0
free1_1	13	10	3	0	0
free1_2	13	11	2	0	0
free1_3	13	10	3	0	0
free1_4	13	11	2	0	0
free2_1	16	10	6	0	0
free2_2	16	10	6	0	0
free2_3	16	10	6	0	0
free2_4	16	10	6	0	0
free3	16	10	6	0	0
free4	0	0	0	0	0
inv1	9	9	0	0	0
inv2	12	12	0	0	0
inv3	0	0	0	0	0
inv4	0	0	0	0	0
rmv	15	14	1	0	0
add	15	14	1	0	0
rmv_end	2	2	0	0	0
add_end	2	2	0	0	0
inv5	3	0	3	0	0
inv6	1	1	0	0	0
inv7	17	17	0	0	0
inv8	15	15	0	0	0
inv10	10	10	0	0	0
inv9	13	13	0	0	0
inv17	15	14	1	0	0
inv16	15	11	4	0	0
inv11	15	14	1	0	0
inv12	15	11	4	0	0
inv13	15	9	6	0	0
inv15	15	9	6	0	0
inv14	15	9	6	0	0
inv18	15	9	6	0	0

Fig.14 Statistic of proof obligation for M6

图 14 M6 层证明义务统计

© 中国科学院软件研究所 http://www.jos.org.cn

至此,通过逐层精化形成了一个可以描述双向链表的形式化模型.通过各层的不变式验证了内存模型具有连续的性质、不存在毗邻空闲块的性质、内存块分配释放无重叠的性质.具有双向链表的内存块模型,相同大小的内存块组成的链表中内存块都是空闲的,无重叠且唯一,可以按照 TLSF 算法进行快速内存查找和分配释放.模型性质的验证实际上是模型状态改变后要求各个变量仍然使不变式成立,在 Rodin 建模工具中利用系统提供的辅助证明很容易验证不变式是否成立.

3.8 验证结论

模型 M0 将实际物理内存池形式化描述为无限离散的集合,将内存块抽象描述为无限离散的集合,将内存块与实际所对应的内存池描述为内存块到内存池的映射关系.每个内存块映射到内存池的不同部分,且每个块的映射值域交集为空以表示每个块不会重叠.模型 M1 每个内存块逻辑上都是有起始位置和结束位置的,实际情况下是起始地址和末尾地址,两者之间的间距认为是块的大小.模型 M1 在 M0 的基础上进行精化,将整个内存池抽象为一段连续的有具体长度的自然数.并且M0描述的不变式性质在模型1中得到了继承,每一次精化都会继承上一层的不变式.正是通过不断精化的过程,模型才变得更具体、更接近实际情况.

模型 M1~M5 共同描述了一个大小有限内存池,内存池由连续的并且具有固定大小的内存块组成.按是否已被分配,内存块分为空闲块和非空闲块.相邻内存块无重叠,且不可能都是空闲块.内存块按大小组成了块大小和内存块的映射关系,每个内存块对应一个相同大小的空闲内存块.

在精化到模型M6时,引入了链表型数据结构,实现了TLSF算法的形式化描述.通过每层模型的不变式和对每层模型中的事件分类来描述第 2.3 节提出的性质(性质 18~性质 21 涉及到性能的性质并未描述验证),在各层模型操作过程中都有对应验证.

在 Rodin 环境下的证明义务验证情况汇总见表 2.

Table 2 Summary of proof obligations

表 2 证明义务汇总

模型层次	证明义务总数	自动证明	手动证明
M0	36	31	5
M1	62	50	12
M2	35	21	14
M3	37	13	24
M4	16	16	0
M5	4	4	0
M6	221	171	50

4 结束语

本文介绍了一种使用 Event-B 来形式化构建内存管理模型的方法,通过逐层精化构建了具有双链表的复杂数据结构的内存模型,模型模拟了内存使用的 TLSF 算法,可实现内存块的释放与分配,以及合并空闲内存块等操作,利用 Rodin 建模工具验证了模型具有实际内存块的连续、无重叠等性质.未来的工作将按照如图 6 所示的技术流程,继续 M6 层模型精化及后续代码自动生成部分工作.另外,关于算法的实时性问题,还有待于进一步研究,在 Event-B 中很难描述强实时性质,对 Event-B 进行扩展或者转换到其他混成系统下都是一种思路.

References:

[1] Jackson D. A direct path to dependable software. Communications of the ACM, 2009,52(4):78−88. [doi: 10.1145/1498765.1498787]

[2] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1983,26(1):53−56. [doi: 10.1145/357980.358001]

[3] Lamport L. Proving the correctness of multiprocess programs. IEEE Trans. on Software Engineering, 1977,2(3):125−143. [doi: 10.1109/TSE.1977.229904]

[4] Reeves G, Neilson T. The mars rover spirit FLASH anomaly. In: Proc. of the Aerospace Conf. IEEE, 2005. [doi: 10.1109/AERO.2005.1559723]

[5] Krebbers R, Leroy X, Wiedijk F. Formal C semantics: CompCert and the C standard. In: Proc. of the 5th Conf. on Interactive Theorem Proving (ITP 2014). Vienna: Springer-Verlag, 2014. 543−548. [doi: 10.1007/978-3-319-08970-6_36]

© 中国科学院软件研究所 http://www.jos.org.cn

[6] Baumann C, Bormer T. Verifying the PikeOS microkernel: First results in the Verisoft XT Avionics project. In: Proc. of the Doctoral Symp. on Systems Software Verification (SSV 2009). 2009. 20–22.
[7] Yu DC, Hamid NA, Shao Z. Building Certified Libraries for PCC: Dynamic Storage Allocation. Berlin, Heidelberg: Springer-Verlag, 2003. [doi: 10.1007/3-540-36575-3_25]
[8] Saltzer JH, Kaashoek MF. Principle of Computer System Design, An Introduction. Morgan Kaufmann Publishers, 2009.
[9] Wilcox M. I'll do it later: Softirqs, tasklets, bottom halves, task queues, work queues and timers. linux.conf.au, 2003.
[10] Corbet J, Rubini A, Kroah-Hartman G. Linux Device Drivers. 3rd ed., Oreilly and Associates Inc., 2005.
[11] McKusick MK, Neville-Neil GV. The Design and Implementation of the FreeBSD Operating System. Pearson Education, 2004.
[12] Walker BJ, Kemmerer RA, Popek GJ. Specification and verification of the UCLA Unix security kernel. Communications of the ACM, 1980,23(2):118−131. [doi: 10.1145/358818.358825]
[13] Mel G. Understanding the Linux Virtual Memory Manager. Upper Saddle River: Prentice Hall, 2004.
[14] Marco Cesati DP. Understanding the Linux Kernel. 3rd ed., O'Reilly Media, Inc., 2005.
[15] Love R. Linux Kernel Development. 3rd ed., Novel Press, 2005.
[16] Fankhauser G, Conrad C, Zitzler E, Plattner B. Topsy—A teachable operating system [Ph.D. Thesis]. ETH Zürich: Computer Engineering and Networks Laboratory, 2000.
[17] Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. seL4: Formal verification of an OS kernel. In: Proc. of ACM SIGOPS the 22nd Symp. on Operating Systems Principles. ACM, 2009. 207−220. [doi: 10.1145/1629575.1629596]
[18] Klein G. Operating system verification—An overview. Sadhana, 2009,34(1):27–69. [doi: 10.1007/s12046-009-0002-4]
[19] Klein G, Tuch H. Towards verified virtual memory in L4. In: Proc. of the TPHOLS EMERGING TRENDS 2004. Park City, 2004.
[20] Qiao L, Yang MF, Gu B. An embedded operating system design for the Lunar exploration rover. In: Proc. of the 5th IEEE Conf. on SSIRI. 2011. 160−165. [doi: 10.1109/SSIRI-C.2011.39]
[21] Masmano M, Ripoll A, Crespo A, Real J. TLSF: A new dynamic memory allocator for real-time systems. In: Proc. of the ECRTS. 2004. 79−88. [doi: 10.1109/EMRTS.2004.1311009]
[22] Masmano M, Ripoll I, Balbastre P, Crespo A. A constant-time dynamic storage allocator for real-time systems. Real-Time Systems, 2008,40(2):149−179. [doi: 10.1007/s11241-008-9052-7]
[23] Marinak KG, Bradley J, Feehan J, Medeiros H, Patel N. Amiga operating system: A brief discussion about history and specifications. 2014. http://www.amiga.com
[24] Crespo A, Ripoll I, Masmano M, Arberet P, Metge JJ. XtratuM: An open source Hypervisors for TSP embedded systems in aerospace. 2014. http://www.xtratum.org
[25] Open Robot Control Software. 2013. http://www.orocos.org
[26] Abrial JR, Leino R. Mini-Course around Event-B and Rodin: hypervisor. 2011. http://research.microsoft.com/apps/video/default.aspx?id=151665
[27] Abrial JR. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.
[28] Hu YC. Research on dynamic memory management of embedded real-time system ARTs-OS [MS. Thesis]. Wuhan: Huazhong University of Science and Technology, 2010 (in Chinese with English abstract).

附中文参考文献:

[28] 胡雨翠.嵌入式实时系统 ARTs-OS 的动态内存管理研究[硕士学位论文].武汉:华中科技大学,2010.

乔磊(1982－),男,江苏新沂人,博士,高级工程师,CCF 专业会员,主要研究领域为嵌入式操作系统设计,形式化验证.

杨孟飞(1962－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为空间飞行器嵌入式系统,控制系统,总体技术.

谭彦亮(1988－),男,工程师,主要研究领域为嵌入式操作系统设计,形式化验证.

蒲戈光(1978－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为程序分析,软件验证,基于 Web 的工作流建模.

杨桦(1969－),女,研究员,CCF 高级会员,主要研究领域为自动控制,嵌入式操作系统设计.

© 中国科学院软件研究所 http://www.jos.org.cn