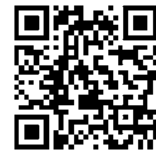
软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(8):2375−2387 [doi: 10.13328/j.cnki.jos.005961]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Coq 的操作系统任务管理需求层建模及验证*



姜菁菁[1], 乔 磊[1,3], 杨孟飞[2], 杨 桦[1], 刘 波[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](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 乔磊, E-mail: fly2moon@aliyun.com

摘 要: 为确保星上操作系统中任务管理设计的可靠性,利用定理证明工具 Coq 对操作系统任务管理模块进行需求层建模及形式化验证.从用户角度,基于星上操作系统任务管理的基本机制,提出一种基于任务状态列表集合的验证框架.在需求层将基本机制进行形式化建模,并在 Coq 中实现.针对建立的需求层模型,提出 6 条与实际星上操作系统任务管理一致的性质并进行验证.给出其中一条性质在 Coq 中的验证过程,结果表明,模型满足该条性质.

关键词: 任务管理;需求层;形式化建模;Coq;形式化验证

中图法分类号: TP306

中文引用格式: 姜菁菁,乔磊,杨孟飞,杨桦,刘波.基于 Coq 的操作系统任务管理需求层建模及验证.软件学报,2020,31(8): 2375−2387. http://www.jos.org.cn/1000-9825/5961.htm

英文引用格式: Jiang JJ, Qiao L, Yang MF, Yang H, Liu B. Operating system task management requirements layer modeling and verification based on Coq. Ruan Jian Xue Bao/Journal of Software, 2020,31(8):2375−2387 (in Chinese). http://www.jos.org.cn/1000-9825/5961.htm

Operating System Task Management Requirements Layer Modeling and Verification Based on Coq

JIANG Jing-Jing[1], QIAO Lei[1,3], YANG Meng-Fei[2], YANG Hua[1], LIU Bo[1]

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

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

[3](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: In order to ensure the reliability of task management design in the operating system on the satellite, the theorem proving tool Coq is used to requirements layer modeling and formal verification of the operating system task management module. From the user point of view for the basic mechanism of on-board operating system task management, this study proposes verification method based on task state list collection. The mechanism process is formalized and implemented in Coq. Six properties are consistent with the task management of the real-world operating system for the established requirements layer model. This article gives a verification process of one of the properties in Coq. The result shows that the model satisfies the properties of the article.

Key words: task management; requirement layer; formal modeling; Coq; formal verification

实时操作系统是航天领域必不可少的基础软件系统.操作系统负责管理计算机的各类硬件资源,操作系统

* 基金项目: 国家自然科学基金(61632005, 61502031); 中国科学院软件研究所计算机科学国家重点实验室开放课题(SYSKF1804)

Foundation item: National Natural Science Foundation of China (61632005, 61502031); Open project of State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences (SYSKF1804)

本文由“面向新兴系统的形式化建模与验证方法”专题特约编辑陈振邦副教授、冯新宇教授、刘志明教授推荐.

收稿时间: 2019-08-31; 修改时间: 2019-11-02; 采用时间: 2019-12-30; jos 在线出版时间: 2020-04-18

© 中国科学院软件研究所 http://www.jos.org.cn

中任何一个小错误都可能导致致命故障.任务管理是操作系统中必不可少的核心功能,任务管理的核心操作为任务调度.在这种实时操作系统中,对其可靠性的保证极其重要,对程序进行形式化建模并对模型进行验证,是目前在高可信计算机领域中一种严格的可靠性保证方法.形式化验证方法是用数学的方法对程序进行形式化规约、建模和验证性质的过程,从而发现程序中的错误和测试方法无法得到的潜在隐患.

操作系统的需求是设计和实现的起点,操作系统的可靠性非常重要.本文对航天领域某实时操作系统的任务管理从用户角度在需求层进行形式化规约和验证,提出了一种基于任务列表集合的验证方法.用的验证语言为 Coq,最终在需求层证明该操作系统的正确性.由于篇幅限制,本文只介绍任务调度和用户操作中最具代表性的任务创建操作的建模和性质验证工作,并在验证过程中对提出的性质中的其中一条进行形式化验证.

1 相关工作

目前,国内外有许多关于操作系统内核上的模型层和代码实现层的形式化验证的研究.模型层的形式化验证和代码实现层的形式化验证大有不同:模型层建模只需描述对模型状态产生影响的需求功能行为和操作语义,不必考虑具体的代码实现细节,不涉及对代码的建模分析;代码层的建模需将实现代码语义转换为验证工具能识别的验证语句,必须涉及实现细节[1,2].Gerwin Klein 等人利用定理证明器 Isabelle/HOL 对 seL4 内核进行模型层和代码层的形式化验证,包括内存管理模块、并发管理模块、IO 管理模块的建模,证明了实现代码的性质和模型层有一致性[1].Gu 等人在代码层建立并发操作系统内核模型并进行分模块验证,该模型是可扩展的体系结构,建模并进行验证的模块包括物理内存管理、虚拟内存管理、共享内存管理、共享队列库、线程管理[3].Feng 等人就验证整个复杂软件系统可靠性的问题,针对不同的模块在验证时使用的验证逻辑可能不同的原因,因此采用对特定模块采用特定验证方法的方式,分别验证可靠性,最后将这些模块连接在一起,验证整个系统的可靠性[4].Fu 等人利用基于依赖保证的模拟方法对并发程序转换进行了验证,验证了并发程序的编译和优化、并发对象的线性化和原子性[5].Xu 等人利用 Coq 对操作系统内核进行建模验证,提出了一种操作系统内核验证框架,并利用该框架证明了μC/OS-II 的优先级反转自由[6].Chen 等人利用 Coq 建立通用设备模型,对可中断的操作系统内核和设备驱动程序进行验证[7,8].Shao 等人利用断言语言对并发线程管理代码进行建模验证,建立的模型为两层框架,允许线程修改自己的线程控制块,提出类 Hoare 逻辑系统验证带硬件中断的抢占式多任务并发系统,并且使用定理证明工具 Coq 证明提出的逻辑系统的可靠性[9,10].Ma 等人提出了结构拆分方法以及分离内存和数据结构形状的方法,实现对某航天器操作系统内核复杂数据结构的形式化定义[11].June 等人在中断并发条件下对任务调度行为证明其正确性,作者只对影响调度决策的部分建模,对模型产生影响的只有两个因素:正在执行的任务和中断处理标志性事件[12].Cao 等人提出了 VST-Floyd 半自动策略以及层次演算方法,该策略在 Coq 工具中展示如何指定、编程、验证和组合模型层,在实际的编程语言中实例化层次演算,VST-Floyd 采用前向验证的思路,用于顺序代码的验证[13].Zou 等人利用验证工具 Coq 对并发文件系统实现细粒度的验证,将文件系统并发操作进行线性化分析,提出一种通过辅助机制用来指定、实现和验证具有原子接口的并发文件系统的框架[14].Wang 等人通过提出一个显示的内存推理机制来推理内存划分在多线程程序运行过程中的变化,验证了线程调度、上下文切换及创建和退出操作的性质,实现了对线程部分功能细粒度的验证[15].Ma 等人在μC/OS-II 上对消息队列机制中涉及的几个内部函数进行代码层形式化规范式编写,证明了共享数据结构的不变性[16].Gu 等人对空间操作系统 SpaceOS 内核状态建模为状态机,描述内核的数据结构并对提出的 8 条性质进行形式化验证,证明了内核的正确性[17].Simon 等人为证明 seL4 实现代码与可执行的规范的精化,提出了一个形式化的 Isabelle/HOL 框架.该框架在代码层完成对 seL4 的高性能 c 实现正确实现可执行规范的形式化的机器检查证明[18].Qiao 等人利用 event-B 工具对航天器内存管理算法和代码进行验证,验证了模型具有实际内存块的连续、无重叠等性质[19].

上面学者对操作系统内核的某些模块形式化验证做了大量工作,但是对任务管理模块的需求层的建模和验证工作不够完善.Shao 等人和 June 等人仅对任务调度行为建模和验证,缺乏对任务管理模块中其他系统操作的验证,如任务创建、任务删除、任务挂起等操作.Tan 利用建模工具 event-B 对任务管理进行建模,由于 event-B

© 中国科学院软件研究所 http://www.jos.org.cn

工具的特性,当针对一个任务创建操作的建模涉及多种任务状态情况时,需定义多个关联函数,增加了建模的复杂性和工作量[20].在对一个涉及多种任务状态的操作进行建模时,Coq 具有更强的表达能力,可以仅用一个函数完成对该操作的建模.因此,本文利用 Coq 对任务管理中涉及的操作系统任务调度等一系列任务管理操作进行准确清晰地形式化建模,并降低了验证工作量.

2 基于 Coq 的任务管理需求层模型建立

本文所建立的模型将上层用户操作以及操作系统中的任务调度模型化,当执行用户操作时,会改变任务控制块内容以及系统的整体状态.除此之外,在满足某些特定条件下,会对操作系统中系统调度进行调用.本文的创新之处为:将所有任务 *TCB* 结构连接成任务 *TCB* 列表,并放入系统整体状态结构中,并且能够实时更新任务 *TCB* 列表,使得任何任务都可以访问其他任务 *TCB* 内容但又不能修改其他任务 *TCB*,在任何操作下的执行都能保证可靠性,以及保证在整个系统运行期间总体的任务 *TCB* 列表与任务 *TCB* 保持一致性.除此之外,系统整体状态结构中包含所有状态的任务名列表,为后续的验证工作提供方便.在本文所建立的模型中,任务调度为不涉及时间片的可抢占调度方式.需求层模型建模涉及任务 *TCB* 内容(用 *TData* 表示)、系统整体状态结构(用 *TList* 表示)和对任务执行的操作(用 *c* 表示),任务 *TCB* 和任务集合状态初始化 *start*.

模型为四元组:(*TCB*,*TList*,*c*,*start*).在本模型中没有考虑内存分配的具体操作,只用内存分配是否成功来表示内存的分配情况.模型中涉及的基本数据类型定义以及操作定义介绍如下.

2.1 基本数据类型定义

模型的任务类型 *Tasks*、任务状态 *TaskStatus*、任务的优先级 *Priority* 是确定下来的内容.我们将其中 *Tasks* 和 *TaskStatus* 用归纳方法定义它们的类型.*Priority* 定义为整数类型,并通过一个判断函数限定 *Priority* 的范围.模型中的任务 *TCB* 用一个包含 *TCB* 属性——任务名、任务状态、任务优先级、延时时间、创建任务标志、删除任务标志、任务挂起标志、任务恢复标志、任务延时标志、任务重启标志、任务调度标志、轮转剩余时间清零标志和更新优先级标志的数据结构 *TData* 来表示.总体任务集合状态用一个包含未创建任务列表、就绪任务列表、执行任务列表、挂起任务列表、阻塞任务列表、延时任务列表、挂起延时任务列表、挂起阻塞任务列表、当前最大优先级、当前最大优先级对应的任务列表以及所有任务 *TCB* 列表的数据结构 *TList* 表示.在 Coq 中,定义具体数据类型如下所示:

定义 1(任务集(多个不同任务类型的集合)). 在模型中将任务类型设置为 21 个.这 21 个任务的任务状态为:1 个正在执行的任务、3 个未创建的任务、2 个就绪态的任务、3 个延时态任务、3 个挂起态任务、3 个阻塞态任务、3 个挂起延时态任务、3 个挂起阻塞态任务.未创建任务、就绪态任务、延时态任务、挂起态任务、阻塞态任务、挂起延时态任务、挂起阻塞态任务的优先级分别大于、等于和小于正在执行任务的优先级.2 个就绪态任务优先级分别小于和等于正在执行任务的优先级.由于初始化时正在执行的任务肯定为最高优先级的任务,所以不存在就绪任务优先级大于正在执行任务的情况.这些任务的定义模拟了模型任务集合的一般状态,若新加入没有定义好的任务,则需将该新任务添加到 Tasks 中并改变初始化的系统整体状态集合中.这些任务的定义代表系统初始化后的一般状态(初始化后一般状态见第 2.3.2 节).在 Coq 中定义如下:

Inductive *Tasks*: Type:=
|*t*1: *Tasks* |*t*2: *Tasks* |*t*3: *Tasks* |*t*4: *Tasks* |*t*5: *Tasks* |*t*6: *Tasks* |*t*7: *Tasks* |*t*8: *Tasks* |*t*9: *Tasks* |*t*10: *Tasks*|*t*11: *Tasks* |*t*12: *Tasks* |*t*13: *Tasks* |*t*14: *Tasks* |*t*15: *Tasks* |*t*16: *Tasks* |*t*17: *Tasks* |*t*18: *Tasks* |*t*19: *Tasks*|*t*20: *Tasks* |*t*21: *Tasks*.

*t*1~*t*21 是所有任务的 id,这些任务在使用之前必须被分配内存.

定义 2(任务状态集(多种任务状态类型的集合)). 在模型行中涉及 8 种任务状态类型,在 Coq 中定义所有任务状态如下:

Inductive *TaskStatus*: Type:=
|*uncreat*: *TaskStatus* |*executing*: *TaskStatus* |*ready*: *TaskStatus* |*delay*: *TaskStatus* |*block*: *TaskStatus*|*suspend*: *TaskStatus* |*suspend_delay*: *TaskStatus* |*suspend_block*: *TaskStatus*.

© 中国科学院软件研究所 http://www.jos.org.cn

分别表示未创建、执行、就绪、延时、阻塞、挂起、挂起延时和挂起阻塞 8 种任务状态类型.

定义 3(任务优先级集合(多种任务优先级类型的集合)). 本模型支持任务优先级的个数为 64 个:*Priority*= 0...63.在 Coq 中定义时将优先级定义为无符号整数类型并通过判断函数 *pri_ritional* 判断 *Priority* 是否为 0~63 的整数,任务优先级定义如下:

Definition *priority*:=*nat*.

在本模型中,任务优先级的数值越大,任务优先级越低.

定义 4(单个任务 *TCB* 结构(单个任务控制块数据结构)). 任务 *TCB* 内容包含一个任务的所有私有属性. *TCB* 中包含的这些标志通过定义一个数据结构来实现每个任务属性的管理.在 Coq 中定义如下:

Record *TData*:=

mkTData{

TL: *Tasks*; *TS*: *TaskStatus*; *TP*: *priority*; *TDtime*: *nat*; *creatFlag*: *nat*; *deleteFlag*: *nat*; *suspendFlag*: *nat*; *resumeFlag*: *nat*; *delayFlag*: *nat*;*restartFlag*: *nat*; *scheduleFlag*: *nat*; *clearTickFlag*: *nat*; *updatePriorityFlag*: *nat*;}.

其中,*TL*∈*Tasks* 表示任务名,*TS*∈*TaskStatus* 表示该任务的状态,*TP*∈*Priority* 为任务的优先级,*TDtime*∈*nat* 表示延时时间.*creatFlag*∈*nat* 表示创建任务的标志:如果 *creatFlag*=1,表示可以创建任务,当创建完成后,*creatFlag* 要置为 0;*creatFlag*=0 表示任务已经创建,不能够重新创建.*deleteFlag*∈*nat* 表示删除任务的标志:如果 *deleteFlag*=1,表示任务不是未创建状态,可以执行删除操作,删除操作完成后要置为 0;如果 *deleteFlag*=0,表示任务为未创建状态,不可以执行删除操作.*suspendFlag*∈*nat* 为任务挂起标志:*suspendFlag*=1 表示可以执行挂起操作,挂起操作执行后要置为 0;如果 *suspendFlag*=0,表示不可以执行挂起操作.*resumeFlag*∈*nat* 为任务恢复标志:如果 *resumeFlag*=1,表示可以执行任务恢复操作,执行完任务恢复操作后置为 0;*resumeFlag*=0 表示不可以执行任务恢复操作.*delayFlag*∈*nat* 为任务延时标志:*delayFlag*=1 表示可以执行任务延时,在任务延时操作执行后置为 0;*delayFlag*=0 表示不可以执行任务延时操作.*restartFlag*∈*nat* 为任务重启操作:*restartFlag*=1 表示可以执行任务重启操作,在执行完任务重启后置为 0;*restartFlag*=0 表示不可以执行任务重启操作.*scheduleFlag*∈*nat* 为任务调度标志:*scheduleFlag*=1 表示可以执行任务调度,在执行完任务调度操作之后置为 0;*scheduleFlag*=0 表示不可以执行任务调度操作.*clearTickFlag*∈*nat* 为轮转剩余时间清零标志:*clearTickFlag*=1 表示可以将轮转剩余时间清零,清零完成后置为 0;*clearTickFlag*=0 表示不可以将轮转剩余时间清零.*updatePriorityFlag*∈*nat* 为更新优先级标志:*updatePriorityFlag*=1 表示可以更新优先级,更新完成后置为 0;*updatePriorityFlag*=0 表示不可以更新优先级.

定义 5(系统整体状态集合(系统整体状态数据结构)). 系统整体状态集合(*TList*)包含模型中所有任务状态列表的集合、记录最高优先级和最高优先级对应的任务名列表以及所有任务 *TCB* 列表.*TList* 中包含各个任务状态的列表,用来记录每一个任务所属的状态列表,每当有任务更改其 *TCB* 信息时,都会更新任务状态列表,任务状态列表的作用为对模型进行验证时方便统计,例如统计所有任务列表的总数应为初始状态定义的任务个数,并且任何两个任务状态列表都不包含相同任务名,保证每个任务只有一个任务状态.*TList* 中包含的最高优先级和最高优先级对应的任务名列表,用来判断在经过某任务操作后是否执行任务调度.例如,最高优先级大于正在执行的任务就会触发任务调度.*TList* 中包含的任务 *TCB* 列表用来实时记录所有任务的 *TCB* 信息,保证实时更新任务 *TCB* 信息.整体的 *TList* 能够保证在进行操作时能够获得正确的任务 *TCB* 信息以及在涉及任务调度时能够获得正确的上下文任务.

在 Coq 中定义系统整体状态数据结构 *TList* 如下:

Record *TList*:=

mkTList{

uncreatList: *listTasks*; *executingList*: *listTasks*; *readyList*: *listTasks*; *delayList*:*listTasks*; *blockList*: *listTasks*; *suspendList*: *listTasks*; *suspend_delay_List*: *listTasks*; *suspend_block_List*: *listTasks*; *TminP*: *priority*; *TminPL*: *listTasks*; *TminPTL*: *list_TData*;}.

其中,*uncreatList* 为所有任务中未创建任务的任务名列表;*executingList* 为所有任务中正在执行任务的任务名列表,理论上,在已经初始化所有任务并且正在执行的任务只有一个,则不论进行何种操作,正在执行的任务都只有一个;*readyList* 为所有任务中就绪任务的任务名列表;*delayList* 为所有任务中睡眠任务的任务名列表;*blockList* 为所有任务中阻塞的任务名列表;*suspendList* 为所有任务中挂起的任务名列表;*suspend_delay_List* 为所有任务中睡眠挂起的任务名列表;*suspend_block_List* 为所有任务中阻塞挂起的任务名列表;*TminP* 为当前就绪列表中最大优先级;*TminPL* 为最大优先级 *TminP* 对应的任务列表;*TminPTL* 为所有任务的 *TCB* 列表.

2.2 具体任务操作定义

本文要验证的某实时操作系统的特点为:每个任务周期性执行,支持基于优先级的可抢占任务调度.用户对任务的操作包括任务创建、任务删除、任务挂起、任务延时、任务阻塞、任务调度、任务恢复、任务重启的 8 个操作.任务管理能够实现这些用户操作.在我们验证的操作系统中要求任务具有就绪、未创建、运行、阻塞、睡眠、挂起、阻塞挂起、睡眠挂起这 8 个状态.8 种状态之间的转换图如图 1 所示.

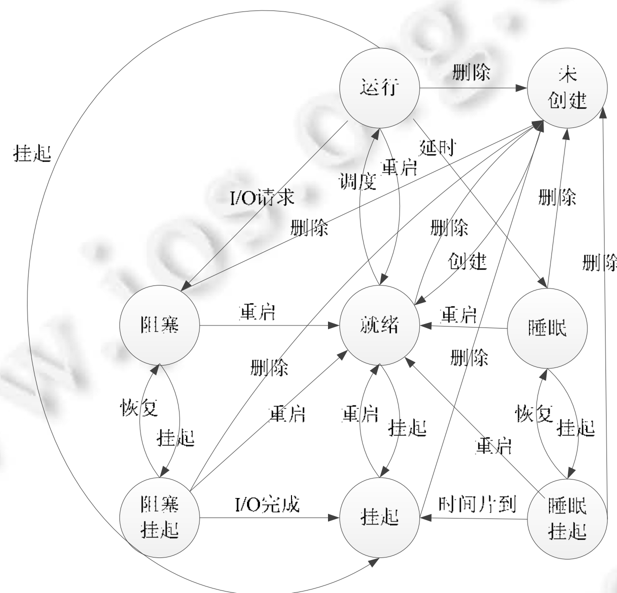


Fig.1 Task conversion process

图 1 任务转换过程

2.2.1 任务调度

本操作系统中的调度为基于优先级的可抢占的调度策略.用户是无法看到任务调度过程的.基础的基于优先级的任务调度为将最高优先级就绪任务转换为运行态,这种情况下,在执行任务调度之前是没有任何任务正在运行的.这个任务调度过程为选择就绪态任务中最高优先级任务运行,将该任务的状态修改为执行态,更新就绪态任务的最高优先级任务.基于优先级的可抢占任务调度不同于不可抢占式的任务调度,这种调度在执行任务调度之前可以有任务正在运行,当就绪态任务列表中最高优先级任务的优先级大于正在执行任务的优先级时,任务调度可以将 CPU 等资源从正在执行的任务中抢占过来给就绪态任务中的最高优先级任务.该调度过程为将正在执行任务修改为就绪态,将最高优先级任务状态修改为运行态,更新就绪任务中最高优先级任务.

在模型中,用一个函数表示任务调度过程.任务调度函数只针对总体状态列表中当前最大优先级任务进行操作,当有新任务被创建或已有任务被删除、挂起、阻塞等操作会使整个系统状态发生变化,并且可能会触发任务调度.任务调度函数的输入参数有两个,分别为单个任务 *TCB* 内容 *TData*(设为 *m*)、系统整体状态集合 *TList*(设为 *t*),输出参数为 3 个,分别为执行某操作后的任务 *TCB* 内容、执行某操作后的系统整体状态集合和是

© 中国科学院软件研究所 http://www.jos.org.cn

否执行成功操作的 *bool* 类型标志.

根据单个任务 *TCB* 中任务的执行状态是否为 *ready* 以及单个任务 *TCB* 中优先级是否大于正在执行任务的优先级,将任务调度分为 4 种情况.用 Coq 形式化定义任务调度过程如下.

Definition *scheduleTasks*(*m*:*TData*)(*t*:*TList*):(*Tdata***TList***bool*):=(*任务调度操作*)

match (*TS m*,*scheduleFlag m*,*isnil_Task*(*remove_one*(*TL m*)(*TminPLt*)),

isnil_Task(*executingList t*)) **with**

1. |(*ready*,1,true,true)⇒(*trans_TD*(*m*)(*executing*)(0),*trans_TL*(*add*(*TL m*)(*executingList t*))(*remove_one*(*TL m*)(*readyList t*))(*finmin*(*remove_TData m*(*TminPTL t*)))((*min_list*(*finmin* (*remove_TData m*(*TminPTL t*)))(*remove_TData m*(*TminPTL t*))))(*add_TData*(*trans_TD*(*m*)(*executing*)(0))(*remove_TData m*(*TminPTL t*))), true) (*情况 1*)
2. |(*ready*,1,false,true)⇒(*trans_TD*(*m*)(*executing*)(0),*trans_TL*(*add*(*TL m*)(*executingList t*))(*remove_one*(*TL m*)(*readyList t*))(*TminP t*)(*remove_one*(*TL m*)(*TminPL t*))(*add_TData*(*trans_TD*(*m*)(*executing*)(0))(*remove_TData m* (*TminPTL t*))),true) (*情况 2*)
   |(*ready*,1,true,false)⇒
   match *min_pri*(*TP m*)(*TP*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))) **with**
3. |true⇒(*trans_TD*(*m*)(*executing*)(0),*trans_TL*(*add*(*TL m*)*nil*)(*add*(*TL*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*)))(*remove_one*(*TL m*)(*readyList t*)))(*finmin*(*remove_TData m*(*add_TData*(*change*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))(*ready*))(*remove_TData*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))(*TminPTL t*)))))(*min_list*(*finmin*(*remove_TData m*(*add_TData*(*change*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))(*ready*))(*remove_TData*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))(*TminPTL t*)))))(*remove_TData m*(*add_TData*(*change*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))(*ready*))(*remove_TData*(*map*1(*fstTasks*(*executingList t*))(*TminPTL t*))(*TminPTL t*)))))(*add_Tdata*(*trans_TD* (*m*)(*executing*)(0))(*remove_TData m*(*TminPTL t*)),true) (*情况 3*)
4. |false⇒(*m*,*t*,false) (*情况 4*)
   end
5. |(_,_,_,_)⇒(*m*,*t*,false) (*情况 4*)

end (*任务调度操作完成*)

上面的 Coq 代码定义中,*isnil_Task* 判断一个任务该列表是否为空,*remove_one* 表示从任务列表中删除一个任务,*trans_TD* 表示修改任务 *TCB* 的任务状态和任务调度标志位,*trans_TL* 表示修改整体任务状态集合中正在执行任务列表、就绪列表、最高优先级的值、最高优先级任务列表、所有任务 *TCB* 列表,*fin_min* 表示从任务 *TCB* 列表中找到最大优先级,*add_TData* 表示向任务 TCB 列表中添加任务 *TCB*,*remove_TData* 表示从任务 *TCB* 列表中移除一个任务 *TCB*,*map*1 表示从任务 *TCB* 列表中查找已知任务名对应的该任务 *TCB* 的结构.任务调度函数可归纳为 4 种情况,具体的说明如下.

(1) 情况 1

当 *TData* 的任务状态为 *ready* 并且为最高优先级任务、调度任务标志为 1,系统整体状态执行列表为空,系统整体状态集合中最大优先级任务列表删除任务 *m* 后列表为空时:执行任务调度后,将 *m* 的状态改为 *executing*,调度任务标志改为 0,修改总体任务集合 *t* 的内容,将 *t* 的 *readyList* 列表删除 *m* 的任务名,将 *m* 的任务名加入到 *t* 的 *executingList* 中,将 *t* 的最高优先级、最高优先级对应的任务列表更新、将 *t* 的所有任务 *TCB* 列表删除 *m* 并添加修改后的 *m*.

(2) 情况 2

当 *TData* 的任务状态为 *ready* 并且为最高优先级任务、调度任务标志为 1,系统整体状态执行列表为空,系统整体状态集合中最大优先级任务列表删除任务 *m* 后列表不为空时:执行任务调度后,将 *m* 的状态改为

executing,调度任务标志改为 0,修改总体任务集合 *t* 的内容,将 *t* 的 *readyList* 列表删除 *m* 的任务名,将 *m* 的任务名加入到 *t* 的 *executingList* 中,将 *t* 的最高优先级对应的任务列表删除任务 *m* 的任务名、将 *t* 的所有任务 *TCB* 列表删除 *m* 并添加修改后的 *m*.

(3) 情况 3

当 *TData* 的任务状态为 *ready* 且为最高优先级任务、调度任务标志为 1,系统整体执行状态列表不为空,*Tdata* 的任务优先级大于正在执行任务优先级,系统整体状态集合中最大优先级任务列表删除任务 *m* 后列表为空时,发生抢占式调度,具体操作为:将 *m* 的状态改为 *executing*,调度任务标志为 0,将 *t* 的 *readyList* 列表删除 *m* 的任务名,将 *t* 的 *executingList* 列表置空并将 *m* 的任务名加入到 *executingList* 中,将正在执行任务名添加到 *readyList* 列表,更新最高优先级和最高优先级任务名列表,删除 *t* 的所有任务 *TCB* 列表中的 *m* 和正在执行任务,之后添加更新后的 *m* 和更新后的正在执行任务.

(4) 情况 4

除上面情况外,执行任务调度后单个任务控制块内容 *m* 和系统整体状态 *t* 不改变.

2.2.2 任务创建

任务创建是用户操作,在任务创建的过程中会调用任务调度.当用户对一个任务执行创建任务操作,如果该任务已经被创建,则不做任何操作;如果任务没有被创建,则为该任务分配内存,初始化其任务控制块,将该任务状态修改为就绪态并插入就绪列表.之后比较其优先级与就绪列表中最高优先级:若小于最高优先级,则结束完成创建操作;若大于最高优先级,则更改就绪列表最高优先级,并且新的最高优先级大于正在执行任务的优先级,则触发调度完成创建操作;若等于最高优先级,则在就绪列表最高优先级任务中插入新创建的任务.

在 Coq 中任务创建函数的两个参数为任务 *TCB* 内容 *TData*(命名为 *m*),系统整体状态集合 *TList*(命为 *t*).根据单个任务 *TCB* 执行状态以及该任务与正在执行任务的关系将任务创建分为 5 种情况,任务创建的 5 种情况用 Coq 形式化定义如下.

```
Definition creatTask(m:TData)(t:TList):(TData*TList*bool):=(*创建任务操作*)
  match (TS m,pri_r(TP m),creatFlag m,min_pri(TP m)(TminP t),min_eqpri(TP m)(TminP t)) with
  |(uncreat,true,1,true,false)⇒
    match min_pri(TP m)(TP(map1(fstTasks(executingList t))(TminPTL t))) with
1.  |true⇒(trans_TD1 m ready 0,scheduleTasks(trans_TD1 m ready 0)(trans_TL1(remove_one(TL m)
    (uncreatList t))(add(TL m)(readyList t))(TP m)(TL m::nil)(add_TData(trans_TD1 m ready 0)
    (remove_TData m(TminPTL t))),true))   (*情况 1*)
2.  |false⇒((trans_TD1 m ready 0),trans_TL1(remove_one(TL m)(uncreatList t))(add(TL m)(readyList t))
    (TP m)(TL m::nil)(add_TData(trans_TD1 m ready 0)(remove_TData m(TminPTL t))),true)   (*情况 2*)
  end
3.  |(uncreat,true,1,false,true)⇒((trans_TD1 m ready 0),trans_TL1(remove_one(TL m)(uncreatList t))
    (add(TL m)(readyList t))(TminP t)(cons(TL m)(TminPL t))(add_TData(trans_TD1 m ready 0)
    (remove_TData m(TminPTL t))),true)   (*情况 3*)
4.  |(uncreat,true,1,false,false)⇒((trans_TD1 m ready 0),transTL1(remove_one(TL m)(uncreatList t))
    (add(TL m)(readyList t))(TminP t)(TminPL t)(add_TData(trans_TD1 m ready 0)(remove_TData m
    (TminPTL t))),true)   (*情况 4*)
5.  |(_,_,_,_,_)⇒(m,t,false)   (*情况 5*)
end   (*创建任务操作完成*)
```

上面的 Coq 代码中,*pri_r* 用来判断函数优先级是否在 0~63 之间,*min_pri* 判断第 1 个参数优先级的值是否小于第 2 个参数优先级,*min_eqpri* 判断两个参数优先级是否相等,*trans_TD*1 用来修改任务 *TCB* 中的任务状态和创建标志位,*trans_TL*1 用来修改整体状态集合中的未创建列表、就绪列表、最高任务优先级、最高优先级

© 中国科学院软件研究所 http://www.jos.org.cn

任务列表、所有任务 TCB 列表,*add_TData* 表示向任务 *TCB* 列表中添加任务 *TCB*,*remove_TData* 表示从任务 *TCB* 列表中移除一个任务 *TCB*.定义的任务创建函数可归纳为 5 种情况,具体的说明如下.

(1) 情况 1

新创建任务的优先级大于正在执行任务的优先级,触发调度.更新的具体内容为:修改 *m* 的任务状态为 *ready*,修改 *m* 的任务创建标志为 0,*t* 的未创建任务名列表删除 *m* 对应的任务名、向 t 的就绪任务名列表中添加 *m* 对应的任务名、更新 *t* 的最高优先级为 *m* 的优先级,更新最高优先级任务为 *m* 的任务名、将 *t* 的总体 *TCB* 列表中的 *m* 删除并添加更新后的 *m*.然后,以更新后的 *m* 和更新后的 *t* 作为参数触发调度函数完成任务创建.

(2) 情况 2

新创建的任务 *m* 优先级大于系统整体状态集合 *t* 中的最高优先级,并且新创建任务优先级不大于正在执行任务的优先级,则经过创建函数操作后,更新任务 *TCB* 内容和系统整体状态集合.更新的具体内容为:修改 *m* 的任务状态为 *ready*,修改 *m* 的任务创建标志为 0,*t* 的未创建任务名列表删除 *m* 对应的任务名、向 *t* 的就绪任务名列表中添加 *m* 对应的任务名、更新 *t* 的最高优先级为 *m* 的优先级,更新最高优先级任务为 *m* 的任务名、将 *t* 的总体 *TCB* 列表中的 *m* 删除并添加更新后的 *m*,完成任务创建.

(3) 情况 3

新创建的任务 *m* 的优先级和总体任务集合 *t* 的最高优先级相等,则经过任务创建后,更新任务 *TCB* 内容和总体任务集合.更新具体实现为:将 *m* 的任务状态设置为 *ready*,将 *m* 的任务创建标志修改为 0,将 *t* 的未创建任务名列表删除 *m* 的任务名,将 *t* 的就绪任务名列表添加 *m* 的任务名,将最高优先级任务名列表添加 *m* 的任务名,将 *t* 的总体 *TCB* 列表中的 *m* 删除并添加更新后的 *m*,完成任务创建.

(4) 情况 4

新创建的任务 *m* 的优先级小于总体任务集合 *t* 的最高优先级,则更新的过程为:将 *m* 的任务状态设置为 *ready*,将 *m* 的任务创建标志设置为 0,将 *t* 的未创建任务名列表删除 *m* 的任务名,将 *t* 的就绪任务名列表添加 *m* 的任务名,将 *t* 的总体 *TCB* 列表中的 *m* 删除并添加更新后的 *m*,完成任务创建.

(5) 情况 5

除上述 4 种情况外,执行创建操作后单个任务 *m* 和系统整体状态 *t* 都不改变.

2.2.3 初始化任务 *TCB* 以及系统整体状态

本文给出 21 个任务的初始化单个任务 *TCB* 内容 *s*1~*s*8_3,代表了任务初始化的一般情况.其中:*s*1 任务代表正在执行中的任务;任务 *s*2_1~*s*2_3 分别代表未创建任务,且优先级分别大于、等于、小于执行态任务的优先级;*s*3_1,*s*3_2 分别代表就绪态任务,且优先级分别等于、小于执行态任务优先级;*s*4_1~*s*4_3 分别代表睡眠态任务,且优先级分别大于、等于、小于执行态任务的优先级;*s*5_1~*s*5_3 分别代表挂起态任务,且优先级分别大于、等于、小于执行态任务的优先级;*s*6_1~*s*6_3 分别代表阻塞态任务,且优先级分别大于、等于、小于执行态任务的优先级;*s*7_1~*s*7_3 分别代表阻塞挂起态,且优先级分别大于、等于、小于执行态任务的优先级;*s*8_1~*s*8_3 分别代表睡眠挂起态,且优先级分别大于、等于、小于执行态任务优先级.并且根据初始化的任务给出整体任务状态 *startlist*,整体任务状态 *startlist* 由系统整体状态数据结构 *TList* 构成.在这个初始化的状态中,正在执行任务是最高优先级任务且只有一个,就绪列表中没有任务优先级高于正在执行任务的优先级,且一个任务只对应一个优先级.在初始化时满足静态性质的要求.具体的 Coq 定义如下.

Definition *s*1:=*mkTData*(*t*1)(*executing*)(8)(0)(1)(1)(1)(1)(1)(0)(1)(1)(1).

Definition *s*2_1:=*mkTData*(*t*2)(*uncreat*)(1)(0)(1)(1)(1)(1)(1)(1)(1)(1)(1).

Definition *s*2_2:=*mkTData*(*t*3)(*uncreat*)(8)(0)(1)(1)(1)(1)(1)(1)(1)(1)(1).

Definition *s*2_3:=*mkTData*(*t*4)(*uncreat*)(9)(0)(1)(1)(1)(1)(1)(1)(1)(1)(1).

Definition *s*3_1:=*mkTData*(*t*5)(*ready*)(8)(0)(0)(1)(1)(1)(1)(1)(1)(1)(1).

Definition *s*3_2:=*mkTData*(*t*6)(*ready*)(10)(0)(0)(1)(1)(1)(1)(1)(1)(1)(1).

Definition *s*4_1:=*mkTData*(*t*7)(*delay*)(1)(8)(0)(1)(1)(0)(1)(1)(1)(1)(1).

© 中国科学院软件研究所 http://www.jos.org.cn

Definition *s*4_2:=*mkTData*(*t*8)(*delay*)(8)(8)(0)(1)(1)(0)(1)(1)(1)(1)(1).

Definition *s*4_3:=*mkTData*(*t*9)(*delay*)(11)(8)(0)(1)(1)(0)(1)(1)(1)(1)(1).

Definition *s*5_1:=*mkTData*(*t*10)(*suspend*)(5)(0)(0)(1)(0)(1)(1)(1)(1)(1)(1).

Definition *s*5_2:=*mkTData*(*t*11)(*suspend*)(8)(0)(0)(1)(0)(1)(1)(1)(1)(1)(1).

Definition *s*5_3:=*mkTData*(*t*12)(*suspend*)(12)(0)(0)(1)(0)(1)(1)(1)(1)(1)(1).

Definition *s*6_1:=*mkTData*(*t*13)(*block*)(1)(0)(0)(1)(1)(1)(0)(1)(1)(1)(1).

Definition *s*6_2:=*mkTData*(*t*14)(*block*)(8)(0)(0)(1)(1)(1)(0)(1)(1)(1)(1).

Definition *s*6_3:=*mkTData*(*t*15)(*block*)(13)(0)(0)(1)(1)(1)(0)(1)(1)(1)(1).

Definition *s*7_1:=*mkTData*(*t*16)(*suspend_block*)(1)(0)(0)(1)(0)(1)(0)(1)(1)(1)(1).

Definition *s*7_2:=*mkTData*(*t*17)(*suspend_block*)(8)(0)(0)(1)(0)(1)(0)(1)(1)(1)(1).

Definition *s*7_3:=*mkTData*(*t*18)(*suspend_block*)(14)(0)(0)(1)(0)(1)(0)(1)(1)(1)(1).

Definition *s*8_1:=*mkTData*(*t*19)(*suspend_delay*)(1)(8)(0)(1)(0)(0)(1)(1)(1)(1)(1).

Definition *s*8_2:=*mkTData*(*t*20)(*suspend_delay*)(8)(8)(0)(1)(0)(0)(1)(1)(1)(1)(1).

Definition *s*8_3:=*mkTData*(*t*21)(*suspend_delay*)(15)(8)(0)(1)(0)(0)(1)(1)(1)(1)(1).

Definition *startlist*:=*mkTList*(*t*2::(*t*3::(*t*4::*nil*)))(*t*1::*nil*)(*t*5::(*t*6::*nil*))(*t*7::(*t*8::(*t*9::*nil*)))(*t*13::(*t*14::(*t*15::*nil*))) (*t*10::(*t*11::(*t*12::*nil*)))(*t*19::(*t*20::(*t*21::*nil*)))(*t*16::(*t*17::(*t*18::*nil*)))(8)(*t*5::*nil*)(*cons*1 *s*1 (*cons*1 *s*2_1(*cons*1 *s*2_2 (*cons*1 *s*2_3(*cons*1 *s*3_1(*cons*1 *s*3_2(*cons*1 *s*4_1(*cons*1 *s*4_2(*cons*1 *s*4_3(*cons*1 *s*5_1(*cons*1 *s*5_2(*cons*1 *s*5_3 (*cons*1 *s*6_1(*cons*1 *s*6_2(*cons*1 *s*6_3(*cons*1 *s*7_1(*cons*1 *s*7_2(*cons*1 *s*7_3 *nil*1)))))))))))))))))).

2.3 性质证明

对于第 2 节建立的任务管理模型,对于如何保证模型与实际操作系统的需求层模型保持一致性问题,本文通过研究实际操作系统的性质,针对本文的模型提出几条与实际操作系统一样的性质.若通过定理证明工具Coq得出这几条性质在本文建立的模型中能够正确的被证明,那么说明本文建立的模型是正确的.本文提出的6条性质包含2条静态定义的性质和4条需证明的动态性质,静态定义的性质是在建模过程中已经定义好的满足要求的性质,动态性质为在模型运行过程中需要动态证明的性质.

2.3.1 定义时已经满足的静态性质

在建立模型时,对于任务优先级,在数据结构中只设置一个优先级,对任务执行何种操作都能保证一个任务只对应一个优先级.任务数据结构中设置状态标志位的目的为标识任务当前所处的状态,对任务执行一个操作时,首先判断标志位状态.例如对一个就绪态任务执行创建操作,该任务当前未创建标志位 *creatFlag*=0 时表示已经创建完成,再次执行创建操作不成功.

本文所提到的2条静态性质如下.

(1) 一个任务只能定义一个优先级,以任务创建为例进行形式化定义如下:

Theorem *priority_equal*: **forall** (*x*:*TData*)(*y*:*TData*),

y=*firstarg*(*creatTask x startlist*)→*TP x*=*TP y*.

(*对任何任务进行创建操作,操作执行完成后,该任务优先级与操作执行前的优先级都相同*)

上面定理中的函数*firstdarg*表示返回类型为(*TData***TList***bool*)的第1个参数的值,类型即为*TData*.上面的定理表示对任何任务进行创建操作,该任务操作完成后优先级与操作执行前优先级都相同.

(2) 模型不存在重复同一操作的情况,在定义时通过设置每一个任务执行某操作时的标志来保证这种情况不存在,以任务重启为例进行形式化描述如下:

Theorem *unchange_Flag*: **forall** (*x*:*TData*)(*z*:*TData*),

z=*first arg*(*restartTask*(*firstarg*(*restartTask x startlist*))(*secondarg*(*restartTask x startlist*)))

→*equal_Flag*(*firstarg*(*restartTask x startlist*))(*z*)=true.

(*对任何任务执行一次重启操作和执行两次重启操作后该任务标志位都相同*)

© 中国科学院软件研究所 http://www.jos.org.cn

上面定理中,*z* 表示对同一执行两次重启操作得到的新任务状态,函数 *equal_Flag* 的作用为比较两个类型为 *TData* 的任务所有任务标志是否完全相同.该定理说明对任何任务,对其执行两次重启操作和执行一次重启操作后任务的所有标志位都相同,表明模型不存在重复同一操作的情况,即使存在,该任务状态也不改变.

上述几种定义时已经满足的静态性质在建立模型中已经被实现,对 *TCB* 内容数据结构以及初始化时已经被定义好,因此能够保证模型在整个运行过程中都满足这些性质.

2.3.2 需证明的动态性质

动态进行任务管理时需检验的性质.

性质 1. 无论对任何队列进行何种操作,一个任务只能对应一个任务状态,即任何两个任务列表都不相交.

本文以任务创建操作为例验证对 *s*2_2 任务执行创建操作后每两个状态列表都不相交.在 Coq 中对性质 1 提出的定理如下.

Theorem *list_uncross*: **forall** *x*:*TList*, *x*=*secondarg*(*creatTask s*2_1 *startlist*)→
NoDup(*uncreatList x*)(*executingList x*)(*readyList x*)(*delayList x*)(*blockList x*)(*suspendList x*)(*suspend_delay_List x*)(*suspend_block_List x*)=false. (*所有任务状态列表两两之间都没有相交任务*)

NoDup 的作用是判断各个任务状态列表是否有相交项.*NoDup* 的具体定义如下:

```
Definition NoDup(a b c d e f g h:listTasks):bool:=
  match (intersect a(b++c++d++e++f++g++h)) with
  |true⇒true
  |false⇒
  match (intersect b(c++d++e++f++g++h)) with
    |true⇒true
    |false⇒
      match (intersect c(d++e++f++g++h)) with
        |true⇒true
        |fasle⇒match (intersect d(e++f++g++h)) with
          |true⇒true
          |false⇒match (intersect e(f++g++h)) with
            |true⇒true
            |false⇒match (intersect f(g++h)) with
              |true⇒true
              |false⇒match (intersect g h) with
                |true⇒true
                |false⇒false
  end  end  end end  end end end
```

上面定理和定义中,符号“++”表示连接两个任务列表,函数 *intersect* 判断两个列表是否包含相同的元素,返回值为 *bool* 类型,包含相同元素返回 true;否则返回 false.上面定理中的函数 *secondarg* 表示返回类型为(*TData***TList***bool*)的第 2 个参数的值,类型即为 *TList*.上面定理中的函数 *length* 表示计算列表中总共有的元素个数.上述的函数在后面其他性质中是同样的含义.

上面的定理给出:通过对 *s*2 进行创建操作,判断操作后的整体状态列表中所有任务状态列表是否有相交的元素,通过上面的证明过程,得到每两种任务状态列表都不含相同元素.同理,对其他任务进行其他用户操作都能够证明上面定理,本文不做叙述.

性质 2. 任何操作过后,正在执行的任务永远只有一个.

本文以任务创建为例对 *s*2_1 执行创建操作,在 Coq 中对性质 2 提出的定理如下:

© 中国科学院软件研究所 http://www.jos.org.cn

Theorem *executing_account*: **forall** *x y*: *TList*, *x*=*secondarg*(*creatTask s*2_1 *startlist*)→*length*(*executingList x*)=1. (**length* 计算列表长度*)

性质 3. 无论做何种操作,任务列表中任务的总数都等于初始化的任务总个数.

本文以任务延时为例对 *s*1 执行延时操作,在 Coq 中对性质 3 提出的定理如下:

Theorem *list_invari_account*5: **forall** *x*: *TList*, *x*=*secondarg*(*resumeTask s*1 *startlist*)→*length*(*uncreatList x*)+*length*(*readyList x*)+*length*(*executingList x*)+*length*(*delayList x*)+*length*(*blockList x*)+*length*(*suspendList x*)+*length*(*suspend_delay_List x*)+*length*(*suspend_block_List x*)=21.

(*所有类型任务列表相加总数为初始化的任务总数*)

性质 4. 在任务调度中,被选择进行任务调度的任务永远为就绪态任务集中优先级最高的任务.

本文以任务删除为例对 *s*1 执行删除操作,在 Coq 中对性质 4 提出的定理如下:

Theorem *min_Task*: **forall** *x*: *TList*, *x*=*secondarg*(*deleteTasks s*3 *startlist*)→
compare(*map*1(*fstTasks*(*executingList x*))(*TminPTL x*))*x*=true.

(*compare 函数的功能为比较一个正在执行的任务优先级是否大于就绪任务最高优先级*)

定理中的 *compare* 函数的功能为比较两个任务的优先级大小:若第 1 个任务优先级大于第 2 个任务,则返回 *bool* 类型的 true;否则返回 false.

上面 4 种动态性质在 Coq 中的实现代码由于篇幅太长,本文只针对性质一给出证明过程:在系统整体状态 *startlist* 已经定义好的情况下,进行创建、删除、延时、恢复、重启、挂起操作后每两个状态列表都不相交.

3 基于 Coq 的任务管理需求层模型形式化验证

3.1 形式化验证性质1

在 Coq 中对第 2.3.2 节提出的性质 1 的定理形式化证明如下.

1. *Proof*. (*开始证明*)
2. *intros*. (*将所有变量、公式引入上下文环境中*)
3. *unfold intersect*. (*展开判断两个列表是否含相同元素的函数 intersect 的定义*)
4. *simpl*. (*化简,false=false∧false=false∧false=false∧false=false∧false=false∧false=false∧false=false 的目标*)
5. *tauto*. (*自动化简*)
6. *Qed*. (*证明结束*)

证明过程分为 6 步:第 1 步通过 *Proof* 告知 Coq 工具证明开始;第 2 步用 *intros* 规则将变量公式引入上下文环境中方便后续过程查找变量;第 3 步用 *unfold* 规则对自定义的判断两个列表是否含有相同元素的函数进行展开;第四步用 *simpl* 规则将展开的 *intersect* 函数应用到上下文中并化简得到 false=false∧false=false∧false=false∧false=false∧false=false∧false=false∧false=false 的目标;第五步用 *tauto* 规则对第四步得到的目标自动化简;第六步用 Qed 告知 Coq 证明过程结束,完成一个完整证明.

除了上述的证明过程外,初始化的系统状态中所有任务进行所有 8 种任务操作后,系统都满足提出的 6 条基本性质.

3.2 证明结果

在 Coq 中对模型进行第 2.3.1 节中的 4 条性质的验证,结果表明,在 Coq 中建立的模型满足 4 条性质.具体的代码统计见表 1.

© 中国科学院软件研究所 http://www.jos.org.cn

Table 1 Coq code line statistics

表 1 Coq 代码行统计

描述	Coq 代码行数
任务管理模型	412
验证	3 528
总数	3 940

4 结 论

本文介绍了利用 Coq 对星上计算机操作系统的任务管理模块的需求层建模,模型中涉及任务调度以及任务管理等操作,提出了一种基于任务状态列表集合的验证框架,此框架包含建立的模型和对模型提出 2 条静态定义性质和 4 条需动态证明的性质,并对 4 条需动态证明的性质进行验证.本文对其中一条动态性质形式化验证过程进行描述.验证结果表明,需求层模型性质符合实际操作系统任务管理模型的性质.保证了需求层模型和实际操作系统的一致性.

但是本文建立的模型仍需完善.本文只在静态方面定义模型的基本操作,没有涉及时间片的定义.在后续的工作中将添加进时间片,利用时间自动机动态地建模,并验证模型在基于时间的动态运行中是否仍然满足同实际操作系统任务管理相同的 6 条基本性质.本文的建模没有涉及代码层面的工作,后续工作将对代码层的函数和算法进行建模和验证.

References:

[1] Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engerlhardt K, Kolanski R, Norrish M, Sewell T. seL4: Formal verification of an OS kernel. In: Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles. 2009. 207−220.

[2] Gu RH, Koenig J, Ramananandro T, Shao Z, Wu XN, Weng SC, Zhang HZ, Guo Y. Deep specifications and certified abstraction layers. In: Proc. of the 42nd ACM Symp. on Principles of Programming Languages (POPL 2015). 2015. 595−608.

[3] Gu RH, Shao Z, Chen H, Wu XN, Kim J, Sjoberg V, Costanzo D. CertiKOS: An extensible architecture for building certified concurrent OS kernels. In: Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2016). 2016. 653−669.

[4] Feng XF, Shao Z, Guo Y, Dong Y. Combining domain-specific and foundational logics to verify complete software systems. In: Proc. of the VSTTE 2008. 2008. 54−69.

[5] Liang HJ, Feng XF, Fu M. A rely-guarantee-based simulation for verifying concurrent program transformations. In: Proc. of the 39th ACM Symp. on Principles of Programming Languages (POPL 2012). 2012. 455−468.

[6] Xu FW, Fu M, Feng XF, Zhang XR, Zhang H, Li ZH. A practical verification framework for preemptive OS kernel. In: Proc. of the Int'l Conf. on Computer Aided Verification. Springer-Verlag, 2016. 57−59.

[7] Barras B, Boutin S, Cornes C, Courant J, Coscoy Y, Delahaye D, de Rauglaudre D, Filliâtre JC, Giménez E, Herbelin H, Huet G, Laulhère H, Loiseleur P, Muñoz C, Murthy C, Parent C, Paulin C, Saïbi A, Werner B. The Coq Proof Assistant Reference Manual—Version V6.3. 1999.

[8] Chen H, Wu XN, Shao Z, Lockerman J, Gu RH. Toward compositional verification of interruptible OS kernels and device drivers. In: Proc. of the 37th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2016. 431−447.

[9] Guo Y, Feng XF, Shao Z, Shi P. Modular verification of concurrent thread management. In: Proc. of the 10th Asian Symp. on Programming Languages and Systems (APLAS). 2012. 315−331.

[10] Feng XF, Shao Z, Dong Y, Guo Y. Certifying low-level programs with hardware interrupts and preemptive threads. Journal of Automated Reasoning, 2009,42(2-4):301−347.

[11] Ma D, Fu M, Qiao L, Feng XY. Formal specification and verification of complex kernel data structures. Journal of Chinese Computer Systems, 2019,40(2):359−366 (in Chinese with English abstract).

[12] Andronick J, Lewis C, Matichuk D, Morgan C, Rizkallah C. Proof of OS Scheduling Behavior in the Presence of Interrupt-Induced Concurrency. Berlin: Springer-Verlag, 2016. 52−68.

© 中国科学院软件研究所 http://www.jos.org.cn

[13] Cao QX, Beringer L, Gruetter S, Dodds J, Appel AW. VST-Floyd: A separation logic tool to verify correctness of C programs. Journal of Automated Reasoning, 2018,61(1-4):1−56.

[14] Zou M, Ding HR, Du D, Fu M, Gu RH, Chen HB. Using concurrent relational logic with helper for verifying the AtomFS file system. In: Zou M, Ding HR, Du D, Fu M, Gu RH, Chen HB, *et al*., eds. Proc. of the 27th ACM Symp. on Operating System Principles. Deerhurst Resort, 2019.

[15] Wang HB, Guo Y, Chen YY. Verification multithreaded Code with dynamic thread creation and termination. Journal of Chinese Computer Systems, 2010,31(8):1637−1642 (in Chinese with English abstract).

[16] Ma JB, Fu M, Feng XY. Formal verification of the message queue communication mechanism in μC/OS-II. Journal of Chinese Computer Systems, 2016,37(6):1179−1184 (in Chinese with English abstract).

[17] Gu HB, Fu M, Qiao L, Feng XY. Formalization and verification of several global properties of SpaceOS. Journal of Chinese Computer Systems, 2019,40(1):141−148 (in Chinese with English abstract).

[18] Simon W, Gerwin K, Thomas S, June A, David C, Michael N. Mind the gap: A verification framework for low-level C. In: Berghofer S, Nipkow T, Urban C, Wenzel M, eds. Proc. of the 22nd TPHOLs. LNCS 5674, Springer-Verlag, 2009. 500–515.

[19] 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 with English abstract). http://www.jos.org.cn/1000-9825/5218.htm [doi: 10.13328/j.cnki.jos.005218]

[20] Tan YL, Yang H, Qiao L. Formal modeling and verification of task-management requirement for SpaceOS2 on Event-B. Aerospace Control and Application, 2014,40(4):57−62 (in Chinese with English abstract). [doi: 10.3969/j.issn.1674-1579.2014.04.011]

附中文参考文献:

[11] 马顶,付明,乔磊,冯新宇.复杂内核数据结构的形式化描述和验证.小型微型计算机系统,2019,40(2):359−366.

[15] 王海波,郭宇,陈意云.验证带有线程的动态创建和退出的多线程程序.小型微型计算机系统,2010,31(8):1637−1642.

[16] 马杰波,付明,冯新宇.μC/OS-II 中消息队列通信机制的形式化验证.小型微型计算机系统,2016,37(6):1179−1184.

[17] 顾海博,付明,乔磊,冯新宇.SpaceOS 中若干全局性质的形式化描述和验证.小型微型计算机系统,2019,40(1):141−148.

[19] 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 [doi: 10.13328/j.cnki.jos.005218]

[20] 谭彦亮,杨桦,乔磊.基于 Event-B 的 SpaceOS2 操作系统任务管理需求形式化建模与验证.空间控制技术与应用,2014,40(4):57−62. [doi: 10.3969/j.issn.1674-1579.2014.04.011]

姜菁菁(1996－),女,硕士生,主要研究领域为操作系统设计与验证.

杨桦(1969－),女,研究员,CCF 高级会员,主要研究领域为高可信操作系统,星载容错计算机.

乔磊(1982－),男,博士,研究员,CCF 专业会员,主要研究领域为操作系统模型设计,存储管理,文件系统.

刘波(1977－),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为星载计算机体系结构.

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.

© 中国科学院软件研究所 http://www.jos.org.cn