微内核中断机制的形式化设计与验证

李康杰 钱振江 黄 皓

(南京大学软件新技术国家重点实验室 南京 210046) (南京大学计算机科学与技术系 南京 210046)

摘 要 操作系统的正确性和安全性很难用定量的方法进行描述。形式化方法是操作系统设计和验证领域公认的标准方法。以操作系统对象语义模型(OSOSM)为基础,采用形式化方法对微内核架构的中断机制进行了设计和验证,在自行开发的安全可信操作系统VTOS上加以实现,采用Isabelle/HOL对设计过程进行了形式化描述,对VTOS中断机制的完整性进行了验证,这对操作系统的形式化设计和验证工作起到了一定的借鉴意义。

关键词 形式化设计,形式化验证,微内核,中断,完整性

中图法分类号 TP316 **文献标识码** A

Formal Design and Verification of Interrupt Mechanism Based on Microkernel

LI Kang-jie QIAN Zhen-jiang HUANG Hao

(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210046, China)

(Department of Computer Science and Technology, Nanjing University, Nanjing 210046, China)

Abstract It is difficult to describe the correctness and security of the operate system (OS) by quantitative analysis. Formal method is the acknowledged standard one in design and verification for OS. Based on the operate system object semantics model (OSOSM), we designed and verified the interruption mechanism of microkernel architecture using formal method, which was realized on our self-implemented verified trusted operate system (VTOS). Meanwhile, we used the theorem prover Isabelle/HOL to formally describe the design process, and verify the integrality of the interruption mechanism of VTOS. Our research plays certain referential significance on formal design and verification of OS.

Keywords Formal design, Formal verification, Microkernel, Interrupt, Integrity

1 引言

操作系统(Operating System, OS)是其他计算机软件的基础,具有高安全性和高可靠性的要求。形式化方法是验证OS正确性和安全性的公认方法,因此将形式化方法与OS的设计以及实现结合是非常必要的,这样可以最大限度地保证OS的正确性。

近年来国内外不少学者在这些方面做了很多工作。NICTA机构的Klein G.博士带领的研究小组对seL4进行了形式化的验证[1],系统按照访问控制模型、高层设计、低层设计和系统代码实现层等层次来进行划分,目的在于通过验证各个层次之间的一致性来说明系统的正确性。Yale大学shao带领的Flint项目组[2]采用自行开发的VeriML语言和λHOL^{ind}逻辑系统设计[3]和开发防冲击的微内核OS[4],并提出了OCAP开放逻辑框架来实现OS不同模块的验证过程的结合。中国科技大学的陈意云教授带领的小组采用基于逻辑推理的程序验证方法来提高软件可信的程度,重点解决指针类型在程序验证中的问题[5,6],目的在于实现程序的高度自动化验证。

在OS的设计过程中就采用严格的形式化方法来对系统的设计进行描述,使形式化的设计和验证相结合,基于这样的设计思路,设计并实现了VTOS(Verified Trusted Operating System)。VTOS是按照预定的安全目标而实现的微内核架构OS,并采用Isabelle/HOL[7]定理证明工具对其设计和实现进行了一致性验证,因此VTOS达到了一定的安全等级。

本文主要对VTOS的中断部分的设计和验证进行阐述。这里采用OS对象语义模型OSOSM[8]来形式化描述系统,将系统中各个执行主体作为对象来看待,认为系统状态的转换是系统中各个对象相互作用的结果。

本文第2节阐述中断机制的模型;第3节阐述相关的形式化设计和实现;第4节对中断机制的完整性进行验证;最后对本文的工作进行总结,并对未来的发展方向进行一定的展望。

2 中断机制的模型

在前期的形式化工作中已经提出了OS对象语义模型OSOSM[8],这是一种用于描述OS语义的开放框架。本文对VTOS中断机制模型的描述也将采用OSOSM,同时也采用

到稿日期:2012-05-30 返修日期:2012-08-31 本文受国家高技术研究发展计划(863)(2011AA01A202),江苏省"六大人才高峰"高层次人才项目(2011-DZXX-035)以及江苏省高校自然科学研究项目(12KJB520001)资助。

李康杰(1985—),男,硕士生,主要研究方向为操作系统安全和形式化验证,E-mail:fgggg0000@163.com;钱振江(1982—),男,博士生,讲师,主要研究方向为操作系统安全和形式化验证、嵌入式系统;黄 皓(1957—),男,博士,教授,博士生导师,主要研究方向为系统软件、信息安全。

分层方式来阐述，整个中断机制模型分成3个层次：基本功效层、实现层和优化层。下面分别对这3个层次进行阐述。

2.1 基本功效层

在基本功效层中，主要是定义中断机制的基本功能。在VTOS中，中断主要实现了3个功能：1)中断发生后，保存进程的上下文，即保存相关的寄存器；2)找到对应的中断处理程序，并调用这个中断处理程序进行处理；3)从中断处理程序返回后，选出下一个要运行的进程，并恢复这个进程的上下文。

2.2 实现层

实现层主要阐述基本功效层中的系统功能的具体实现。

I. 实现层论域

实现层的论域对象和类型包括：

1)eflags：标志寄存器类型，里面的元素是其相关的标志位，这些标志位均为bool类型，主要包括：溢出位OF、方向标志位DF、中断许可标志位IF、跟踪标志位TF、符号位标志SF、零位标志ZF、辅助进位标志AF、奇偶位标志PF、无符号数进位标志CF等。

2)regs：寄存器对象的集合，保护的寄存器对象有：gs、fs、es、ds、edi、esi、fp、st、ebx、edx、ecx、eax、pc(即ip)、cs、psw、sp和ss，其中psw的类型为eflags，其他寄存器对象的类型都为整型。

3)proc：进程对象类型，与中断相关的元素主要有：p_nr，表示进程的标示符；p_reg，表示进程的寄存器结合，类型为regs；p_ticks_max，表示进程获得的最大的时间片；p_ticks_left，表示进程剩下的时间片。

4)state：系统状态类型，与中断相关的元素主要有：中断发生和屏蔽标志intr_sig，发生中断时，硬件把这个元素的值变为True，表示中断发生且没有被屏蔽；处理中断号real_intr_id，发生中断时，硬件直接把准备处理的中断的中断号的值赋给这个元素；发生中断号链表intr_id_list，表示在CPU运行期间所有发生的中断对应的中断号组成的链表；可处理中断号链表intr_hander_ids，表示当前可以处理的中断对应的中断号所组成的链表；lost_ticks，记录两次时钟中断处理之间丢失的时间片，这个记录的功能由硬件来完成的，当CPU在处理完一次时钟中断的时候，就把这个数变为0；time，记录时间的元素；proc_list，表示进程链表；ready_procs，表示就绪进程列表；now_proc，表示当前运行的进程；next_proc，表示下一个要运行的进程；prev_proc，表示前一个运行的进程；reg_s，表示当前状态的相关寄存器组的值。

5)intr_handle：中断处理程序的类型，即"state => state"，表示输入一个state类型的对象，经过中断处理函数的处理，返回处理后的系统状态。

6)intr_handle_s：中断(硬件中断)处理程序的集合。对于每一种中断，其对应一个处理程序，这些程序链接成一个链表，各个元素类型为intr_handle，中断号为n的中断对应的中断处理程序为这个链表中的第n个元素。

II. 实现层语义函数

中断机制的每个系统操作都将引起系统由一个状态转换到另一个状态，因此实现层语义函数的类型为"state => state"。下面对中断机制对象语义模型在实现层的语义函数进行阐述。

1)cpu_check_intr：处理器每次执行完一条指令都会检查是否有中断发生，从而决定是继续执行下一条指令还是进行中断处理，cpu_check_intr就是描述这个过程的语义函数。也就是说，在语义上，处理器每次执行完一条指令后都会执行这个语义函数。

2)deal_intr：中断处理的入口语义函数。

3)save_regs：该语义函数保存被中断的进程相关的寄存器。

4)intr_ret：中断处理返回的语义函数，其语义是调度下一个将要运行的进程，恢复其中断前的上下文并开启中断。

5)shed：进程调度语义函数，在当前运行进程的时间片用完时进行进程调度。

6)clock_handler：时钟中断处理的语义函数，其完成的工作是更新当前状态的时间，并将当前运行进程的时间片减少，根据当前运行进程剩下的时间片判断是否需要进行进程调度。

2.3 优化层

在优化层，主要描述为提高实现层中语义函数的效率而引入的辅助语义函数。

1)check_IF：该语义函数检查在某个状态下中断是否被屏蔽掉。

2)find_intr_handlers：该语义函数找出对应的中断处理函数所在的链表。

3)update_proc_list：因为每个状态都包含proc_list域，它是一个包含这个状态中所有进程的链表，所以这个链表也包含这个状态中的元素now_proc。由于now_proc经常被改变，导致proc_list经常要更新，语义函数update_proc_list就是实现这个功能的语义函数。

4)update_ready_proc：与上面的update_proc_list类似，因为每个状态都包含一个ready_proc域，它是一个包含这个状态中所有就绪进程的链表，所以这个链表也包含这个状态中的元素now_proc。由于now_proc经常被改变，导致ready_proc经常要更新，语义函数update_ready_proc就是为实现这个功能而设计的。

3 中断机制的形式化描述

3.1 Isabelle/HOL的符号系统

本节首先介绍一下后面形式化描述和验证过程中将用到的符号系统Isabelle/HOL。

表1 Isabelle/HOL符号表示和语法

nat	自然数类型
bool	布尔类型
record p=	
x::"nat"	结构体类型定义
y::"int"	
fun	全函数定义
primrec	原始递归函数定义
a=>b	函数映射关系
'a list	列表类型('a为元素类型变量)
sort	列表排序函数
member a b	检查元素b是否是列表a中的元素
hd x	返回列表x的头元素
tl x	返回列表x去掉头后剩下的链表
remove1 a b	从列表a中删除元素b
s(| x:=y |)	结构体更新操作
lemma a	定理定义
datatype a	新数据类型构造
apply	利用规则来证明命题

Isabelle/HOL[7]是一种定理证明工具(Theorem Prover),可用于对计算机系统进行验证。Isabelle/HOL采用与函数式编程(Functional Programming)类似的语法规范,支持归纳演算、Lambda演算,以及经典逻辑证明,拥有强大的类型表达能力。

表1列出了后面将用到的Isabelle/HOL中的符号表示和语法。

3.2 中断机制的Isabelle/HOL描述

本节阐述采用Isabelle/HOL对VTOS中断机制的形式化描述。

VTOS中断机制包括对时钟中断、内部中断、外部中断等的处理。限于篇幅,本节仅从时钟中断的角度来阐述。VTOS时钟中断处理语义函数调用关系如图1所示。

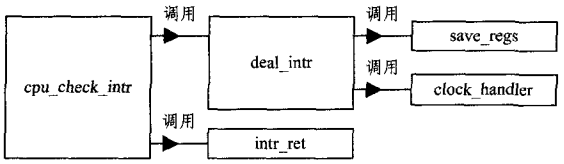

图1 VTOS时钟中断处理语义函数调用关系

下面章节对第2节中部分语义函数在Isabelle/HOL中的描述进行说明。

1)cpu_check_intr:类型为"state => state"。对输入状态s,当s.intr_sig和s.reg_s.psw.IF都为True的时候,说明要处理中断,此时调用函数deal_intr来进行具体的中断处理。同时,把s.intr_id_list中的最小值作为选定要处理的中断号赋给s.real_intr_id,之后把改变后的s作为deal_intr的参数,调用这个函数,返回后再以返回的状态的值作为参数调用intr_ret来恢复被中断的进程(不一定是中断前被打断的进程)。当s.intr_sig和s.reg_s.psw.IF中有一个不为True的时候,直接返回状态s。整个过程如图2所示。

```
fun cpu_check_intr::"state=>state" where
"cpu_check_intr s=(if ((intr_sig s)∧(IF (psw (reg_s s))))
  then
  (intr_ret (deal_intr (s(| real_intr:=(hd (sort (intr_id_list s)))
|))))
else s)"
```

图2 cpu_check_intr函数描述

2)save_regs:类型为"state => state"。这个函数的功能主要是根据输入状态s,将s.reg_s的值赋给s.now_proc.p_reg。同时由于状态中的now_proc域已改变,因此需要更新s.proc_list和s.ready_procs,在此分别调用上面说到的两个辅助函数update_ready_proc和update_proc_list来完成状态s中的这两个元素的更新。整个过程如图3所示。

```
fun save_regs::"state => state" where
"save_regs s=
update_ready_proc
  (update_proc_list
    (s(|now_proc:=((now_proc s)(| p_reg:=(reg_s s) |))|))
    ((now_proc s)(|p_reg:=(reg_s s)|)))
  ((now_proc s)(|p_reg:=(reg_s s)|))"
```

图3 save_regs函数描述

3)deal_intr:类型为"state => state",它调用函数save_regs和clock_handler来完成被中断进程上下文的保存和中断处理过程。整个过程如图4所示。

```
fun deal_intr::"state=>state" where
"deal_intr s=
clock_handler (save_regs s)"
```

图4 deal_intr函数描述

4)clock_handler:时间中断处理语义函数,类型为"state => state",这个函数的功能主要是根据输入状态s,将s.time的值变为s.time+s.lost_ticks+1,即更新s的时间,同时把s.lost_ticks的值变为0,并将s.now_proc.p_tick_left的值与s.lost_ticks+1比较:若前者大于或等于后者,则把s.now_proc.p_ticks_left的值减少s.lost_ticks+1,同时更新s.ready_procs和s.proc_list,之后返回新的s;若前者小于后者,把s.now_proc.p_ticks_left的值变为s.now_proc.p_ticks_max的值,同时更新s.ready_procs和s.proc_list,并调用进程调度函数sched来进行进程调度,以决定s.next_proc的值。整个过程如图5所示。

```
fun clock_handler::"state => state" where
"clock_handler s=
(if p_ticks_left (now_proc s) ≤((lost_ticks s)+1::nat)
then
(sched (update_ready_proc
    (update_proc_list (s(|time:= (time s)+(lost_ticks s)+1::
    nat,lost_ticks:= 0::nat|)) ((now_proc s) (|p_ticks_left:=(p
    _ticks_max (now_proc s)) |)))
    ((now_proc s) (|p_ticks_left:=(p_ticks_max (now_proc s))
    |))))
else
  (update_ready_proc
    (update_proc_list (s(|time:= (time s)+(lost_ticks s)+1::
  nat,lost_ticks:= 0::nat|)) ((now_proc s) (|p_ticks_left:=(p_
  ticks_left (now_proc s))-lost_ticks s |)))
    ((now_proc s) (|p_ticks_left:=(p_ticks_left (now_proc s))-
  lost_ticks s |))))"
```

图5 clock_handler函数描述

5)intr_ret:中断处理返回函数,类型为"state => state",其功能是根据输入状态s,以及s.next_proc.p_reg的值来设置s.reg_s,即恢复s.next_proc的运行环境,同时把s.next_proc的值赋给s.now_proc。整个过程如图6所示。

```
fun intr_ret::"state=>state" where
"intr_ret s=
s(|reg_s:= (p_reg (next_proc s)),now_proc:=next_proc s|)"
```

图6 intr_ret函数描述

4 VTOS中断机制完整性的形式化验证

4.1 VTOS中断机制完整性

在VTOS的设计当中,完整性是一个非常重要的性质,具体来说,要保证系统行为的实现与预期设计的功能需求保持一致。

对VTOS中断机制来说,需要保证中断机制的语义函数符合预定的完整性需求。限于篇幅,本文对VTOS中断机制的进程上下文保护的完整性问题进行验证。

VTOS中断机制的进程上下文保护的完整性定义为:若一个正在运行的进程被中断了,那么在将来某个状态,这个进

程会被选中运行，而且其运行上下文与被中断前的上下文相同，命题表示如下：

$$s, deal\_intr \vdash EF\ (\lambda w.\ w.\ reg\_s = (deal\_intr\ s).\ reg\_s \wedge s.\ now\_proc.\ p\_nr = w.\ now\_proc.\ p\_nr) \quad (1)$$

式中，“state，action ⊢ formula”表示“在状态 state 下执行系统行为 action 之后在将来会满足谓词公式 formula”。“EF”是时序逻辑中“路径将来时刻”的定义表达。式(1)表示：在状态 s 下执行中断处理函数 deal_intr 后，在将来某个时刻存在一个状态 w 会将被中断的进程恢复，即状态 w 的寄存器组和状态 s 的寄存器组相同，并且状态 w 的运行进程 now_proc 的 p_nr 和状态 s 的运行进程 now_proc 的 p_nr 相等。

对于该命题，假设进程调度部分没有问题，即状态 s 的就绪链表里面的进程 p 都会得到运行的机会，也就是说将来存在一个状态 w，其寄存器组与 p 的寄存器组相等并且 w. now_proc. p_nr 和 p. p_nr 相等，即下面的命题为真：

$$forall\ p.\ (member\ s.\ ready\_proc\ p) \Longrightarrow s \vdash EF\ (\lambda w.\ w.\ reg\_s = p.\ p\_reg \wedge w.\ now\_proc.\ p\_nr = p.\ p\_nr) \quad (2)$$

式中，“state ⊢ formula”表示“在状态 state 下在将来会满足谓词公式 formula”。

4.2 完整性形式化验证

本小节对上述的公式进行验证，以说明 VTOS 的设计满足完整性需求。

在验证过程中，涉及到的命题类型包括基础命题 Atom、否定命题 Neg、合取命题 And，以及时序逻辑命题 AX、EF 等，其中基础命题的类型为“state ⇒ bool”，即对状态 s，原子命题 f 成立的时候，fs 的值为 True。定义如图 7 所示。

```
datatype proposition = Atom "state=>bool" |
                       Neg proposition|
                       And proposition|
                       AX proposition|
                       EF propsition
```

图 7 命题类型定义

对于“state，action ⊢ formula”命题的逻辑演算方法，可以转换为对“state ⊢ formula”的证明演算，如图 8 所示。

```
definition cal_action::"state=>(state=>state)=>proposition=>
bool"
("(_,_⊢_)" [81,81,81] 81) where
"s,act ⊢ f ≡ (act s) ⊢ f"
```

图 8 “state，action ⊢ formula”的计算方法

在式(2)中，假设每个就绪列表里面的进程都会得到运行的机会，也就是状态的寄存器组的值和进程的寄存器组的值相等并且这个进程与状态的当前运行进程 now_proc 的 p_nr 相等，为此，可以得到如图 9 所示的、名为 sched_ok 的公理。

```
axioms sched_ok:："member (ready_procs s)p−−>
s ⊢ EF (Atom (λw. ((reg_s w=p_reg p) ∧
    p_nr (now_proc s)=p_nr (now_proc w))))"
```

图 9 sched_ok 公理

这里要证明的是在状态 s 执行中断操作后，在将来会存在状态 w 使得：

w. reg_s = s. reg_s ∧ now_proc. p_nr = s. now_proc. p_nr

在这里称为目标命题，如图 10 所示。

```
lemma "s,deal_intr ⊢ EF (Atom (λw. (reg_s w=reg_s s) ∧
p_nr (now_proc s)=p_nr (now_proc w)))"
```

图 10 目标命题

由于目标命题无法直接证明，因此先引入两个引理：p_nr_equiv 和 interim。p_nr_equiv 这个命题目标是要证明在中断前和中断后的两个状态 now_proc 的 p_nr 相等，命题如下：

s. now_proc. p_nr = (deal_intr s). now_proc. p_nr

命题 interim 的目标是证明在状态 s 执行完 deal_intr 操作后，将来会存在一个状态 w 使得：

w. reg_s = s. reg_s ∧ (deal_intr s). now_proc. p_nr = w. now_proc. p_nr

这样，如果这两个命题成立，那么就可以尝试通过这两个命题来证明目标命题。

命题 p_nr_equiv 的证明比较简单，在 Isabelle/HOL 里面直接用经典逻辑方法 auto 就可以得证，证明如图 11 所示。

```
lemma p_nr_equiv:
"p_nr (now_proc s)=p_nr (now_proc (deal_intr s))"
apply auto
done
```

图 11 引理 p_nr_equiv 的证明

由于引理 interm 无法直接证明，在此需要继续引入辅助命题。我们知道 s. reg_s 的值保存在当前运行的进程中，即 s. reg_s = (deal_intr s). now_proc. p_reg。为此，再引入引理 p_regs_equiv，如下：

s'. now_proc. p_reg = s. reg_s

其中，s' = deal_intr s。

又由于已经有了公理 sched_ok，因此只需要再加一个辅助引理 p_in_ready 就可证明 s'. now_proc 是在 s'的就绪列表里面，也就是如下命题：

member (s'. ready_proc) (s'. now_proc)

在 Isabelle/HOL 中，这两个引理的证明如图 12 所示。

```
lemma p_in_ready:
"member (ready_proc (deal_intr s)) (now_proc (deal_intr s))"
apply auto
done
lemma p_regs_equiv:
"p_reg (now_proc (deal_intr s))=reg_s s"
apply auto
done
```

图 12 引理 p_in_ready 和 p_regs_equiv 的证明

最后，利用引理 p_in_ready 和 p_regs_equiv 以及公理 sched_ok，来证明引理 interm，其证明如图 13 所示。

```
lemma interim:
"s,deal_intr ⊢ EF (Atom (λw. (reg_s w=reg_s s) ∧
p_nr (now_proc (deal_intr s))=p_nr (now_proc w)))"
apply (subst cal_action_def)
apply (insert p_regs_equiv [of s])
apply (insert p_in_ready [of s])
apply (insert sched_ok [of "deal_intr s" "now_proc (deal_intr s)"])
apply (drule mp,assumption)
by (erule subst)
```

图 13 引理 interm 的证明

(下转第 205 页)

[J]. 计算机学报,2010,33(7):1153-1164
[12] 赵新杰,王韬,郭世泽. AES 访问驱动 Cache 计时攻击[J]. 软件学报,2011,22(3):572-591
[13] 赵新杰,王韬,郑媛媛. 针对 SMS4 密码算法的 Cache 计时攻击[J]. 通信学报,2010,31(6):89-98
[14] 赵新杰,郭世泽,王韬,等. 针对 AES 和 CLEFIA 的改进 Cache 踪迹驱动攻击[J]. 通信学报,2011,32(8):101-110
[15] Zhao X J,Zhang F,Guo S Z,et al. MDASCA:An Enhanced Algebraic Side-Channel Attack for Error Tolerance and New Leakage Model Exploitation[C]// Schindler W,Huss S A,eds. COSADE 2012. LNCS 7275,2012:231-248
[16] Percival C. Cache missing for fun and profit[EB/OL]. http://www. daemonology. net/papers/htt. pdf,2005
[17] Aciiçmez O,Brumley B B,Grabher P. New Results on Instruction Cache Attacks[A]// CHES2010[C]. Santa Barbara,USA,2010:110-124
[18] Brumley B B,Hakala R M. Cache-timing template attacks[A]// ASIACRYPT 2009[C]. Tokyo,Japan,2009:667-684
[19] 陈财森,王韬,陈建泗,等. 基于 Cache Missing 的 RSA 计时攻击[J]. 微电子学与计算机,2009,26(5):180-182,186
[20] 郑媛媛,王韬,赵新杰,等. 针对 RSA 密码算法的指令 Cache 攻击方法[J]. 微电子学与计算机,2009,26(2):197-200
[21] Zenner E. A Cache timing analysis of HC-256[A]// SAC 2008[C]. LNCS 5381,2009:199-213
[22] Brumley B B,Hakala R M,Nyberg K,et al. Consecutive S-box Lookups:A Timing Attack on SNOW 3G[A]// ICICS 2010[C]. LNCS 6476,2010:171-185
[23] Leresteux D,Fouque P A,Chardin T. Cache Timing Analysis of RC4[A]// ACNS 2011[C]. LNCS 6715,2011:110-129
[24] Coppersmith D. Finding a small root of a bivariate integer equation;factoring with high bits known[A]// Cryptology EUROCRYPT 96[C]. Saragossa,Spain,1996:178-189
[25] May A. New RSA vulnerabilities using lattice reduction methods[D]. Paderborn,Germany,University of Paderborn,2003
[26] Nguyen P Q,Shparlinski I E. The insecurity of the Digital Signature Algorithm with partially known nonces[J]. Cryptology,2002,15:151-176
[27] Leadbitter P J,Page D,Smart N P. Attacking DSA Under a Repeated Bits Assumption[A]// CHES2004[C]. Boston,USA,2004:141-190
[28] Kuwakado H,Tanaka T. On the security of the ElGamal-Type signature scheme with small paramenters[EB/OL]. http://www. lib. kobe-u. ac. jp/handle_kernel/90001314,1999
[29] Knuth D E. The Art of Computer Programming[M]. Massachusetts:Addison-Wesley,1981
[30] Menezes A J,van Oorscht P C,Vanstome S A. 应用密码学手册[M]. 北京:电子工业出版社,2005
[31] Koc C. Analysis of sliding window techniques for exponentiation[J]. Computers and Mathematics with Application,1995,30(10):17-24
[32] Bryant R E,Hallaron D. 深入理解计算机系统[M]. 北京:机械工业出版社,2011
[33] Babbage S,Catalano D,Cid C. Yearly Report on Algorithms and Keysizes(2011)[R]. http://www. ecrypt. eu. org/documents/D. SPA. 17. pdf,2011

(上接第 200 页)

这样,通过利用引理 interim 和 p_nr_equiv 来证明目标命题,其证明如图 14 所示。

```
lemma "s,deal_intr ⊢EF (Atom (λw. (reg_s w=reg_s s) ∧
    p_nr (now_proc s)=p_nr (now_proc w)))"
apply (insert p_nr_equiv [of s])
apply (insert interim [of s])
by (erule ssubst)
```

图 14 目标命题的证明

在 Isabelle 中最后的证明结果截图如图 15 所示。

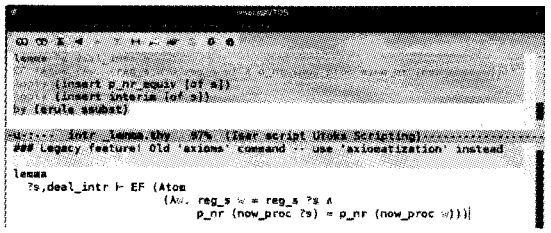

图 15 证明结果

结束语 本文采用形式化方法对 VTOS 的中断机制进行了设计,并采用 Isabelle/HOL 对设计过程进行了描述,对 VTOS 中断机制的完整性问题进行了验证。本文对操作系统形式化设计和验证工作起到了一定的借鉴意义。在具体的设计和验证过程中,不得不承认其工作量非常大,Isabelle/HOL 完整的形式化描述和验证工作在代码量与具体的实现代码上大概为 10∶1 左右。在未来的研究工作中,将考虑采用类型论来实现系统代码的自动构造。

参 考 文 献

[1] Elkaduwe D,Klein G,Elphinstone K. Verified protection model of the seL4 microkernel[R/OL]. http://ertos. nicta. com. au/publications/papers/Elkaduwe_GE_07. pdf,2012-05-25
[2] Yale Flink Project. Website[OL]. http://flink. cs. yale. edu/,2012
[3] Stampoulis A,Shao Z. VeriML:Typed Computation of Logical Terms Inside a Language with Effects[C]// Proc. of the 15th ACM SIGPLAN International Conference on Functional Programming. New York,USA:ACM Press,2010
[4] Shao Z,Ford B. Advanced Development of Certified OS Kernels[D]. New Haven,USA:Yale University,2010
[5] 陈意云,李兆鹏,王志芳. 一种用于指针程序验证的指针逻辑[J]. 软件学报,2010,21(3):414-426
[6] 王振明,陈意云,王志芳. 一个用于指针程序验证的自动定理证明器的设计与实现[J]. 小型微型计算机系统,2010,5:801-806
[7] Nipkow T,Paulson L C. Isabelle/HOL:A Proof Assistant for Higher-order Logic[M]. Berlin,Germany:Springer,2002
[8] 钱振江,刘苇,黄皓. OS 对象语义模型(OSOSM)及形式化验证[J]. 计算机研究与发展,2012(12):2702-2712