面向访问验证保护级的安全VMM形式化原型系统设计和实现

易秋萍[1,2,3] 刘 剑[1,3] 武 术[1]
(中国科学院软件研究所 北京 100190)[1] (中国科学院研究生院 北京 100049)[2]
(计算机科学国家重点实验室 北京 100190)[3]

摘 要 操作系统是计算机软件系统的基础,具有控制逻辑复杂、安全性和可靠性要求高等特点。在国内外高等级安全操作系统的规范和标准中,都提出了对内核进行形式化规范和验证的要求。近年来国内相关研究机构相继开发了满足GB 17859-1999"强制访问控制级"和"结构化保护级"的安全操作系统原型,但对更高级别的安全操作系统的研发尚属空白。在"面向访问验证保护级安全操作系统"课题的研究中,设计并实现了一个基于Haskell的安全VMM原型系统——CASVisor。CASVisor严格定义了系统的形式化规范,可用于指导高性能的C程序的实现,并为形式化的分析和验证打下基础,同时CASVisor具备模拟功能,以便实施基于快速原型的开发方法。

关键词 安全操作系统,VMM,Haskell,Monad,形式化原型
中图法分类号 TP31 **文献标识码** A

Formal Secure VMM Prototype Towards Level Verified Design

YI Qiu-ping[1,2,3] LIU Jian[1,3] WU Shu[1]
(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]
(Graduate University, Chinese Academy of Sciences, Beijing 100049, China)[2]
(State Key Laboratory of Computer Science, Beijing 100190, China)[3]

Abstract Operation systems are the base of computer software system which have complex controlling logics, and their security and reliability are very critical. In almost all of standards of security operating system in the world, formal specification and verification on them are needed. Inrecent years, several system meeting level "mandatory access control" and "structural protection" in GB 17859-1999 were designed and implemented by domestic research agencies. However, systems conformed to higher leveles are still in blank. In this paper, we reported a VMM-based security prototype-CASVisor, a system towards "level verified design", which is designed and implemented in our group. CASVisor has formal definition of the system specification, which can be used to guide high-performance implementation in C programs, and support formal analysis and verification. Moreover, CASVisor can be used as rapid prototype to simulate the system design.

Keywords Secure operating system, VMM, Haskell, Monad, Formal prototype

1 引言

操作系统是计算机软件系统的基础,具有控制逻辑复杂、对安全性和可靠性要求高等特点。高等级安全操作系统是近年来基础软件和软件高可信技术研究的热点,也是国家信息安全等级保护战略的迫切需要。在国内外高等级安全操作系统标准和相关项目的研发中,形式化保证技术都占有重要的地位[1,13]。国际上权威的安全操作系统标准"Trusted Computer System Evaluation Criteria"(TCSEC)[2]明确要求,B2以上的安全操作系统要对其采用的安全策略模型进行形式化规范和分析,验证其是否满足目标安全性质。B3级安全操作系统,除了进行上述工作外还要研发可信计算基(TCB)的描述性顶层规范(DTLS),DTLS需要定义TCB的主要功能以及系统的例外、错误信息、作用(Effects)等;并要分析DTLS和形式化安全策略模型的对应性;分析DTLS和TCB接口的对应性。A1级安全操作系统在B3的基础上提出了更严格的要求,要求在系统开发的生命周期中研发TCB的形式化顶层规范(FTLS),FTLS必须包括系统功能模块、硬件、固件等的形式化抽象定义;并且,如果采用形式化工具,必须严格分析FTLS和安全策略模型之间的对应性关系;分析FTLS和

到稿日期:2010-01-08 返修日期:2010-03-29 本文受中国科学院知识创新工程重要方向项目"面向访问验证保护级的安全操作系统原型系统研发(KGCX2-YW-125)",北京市科技创新项目"安全可信操作系统研制(Z080001020000801)",计算机科学国家重点实验室开放课题"面向高等级安全操作系统的形式化保证技术研究(SYSKF0909)"资助。

易秋萍(1986-),女,硕士生,主要研究方向为程序验证分析、可信系统软件,E-mail:qiuping08@iscas.ac.cn;刘 剑(1976-),男,高级工程师,硕士生导师,主要研究方向为可信系统软件、程序分析及验证、形式化技术;武 术(1986-),男,软件工程师,主要研究方向为操作系统与虚拟化技术。

实现代码之间的对应性。在我国高等级安全操作系统标准中，例如 GB17859-1999[14]，GB/T20272-2006[15,16]等，要求在第四级（结构化保护级）以上的安全操作系统研发中提供形式化保证技术的支持。GB/T20272-2006 明确规定在第四级安全操作系统的研发中要“建立半形式化的完整性安全策略模型”，“并按半形式化功能说明、半形式化高层设计、半形式化低层设计、半形式化对应性说明的要求”进行 TCB 的开发。在第五级（访问验证保护级）安全操作系统的研发中，要求“建立形式化的完整性安全策略模型”，“并按形式化功能说明、形式化高层设计、形式化低层设计、形式化对应性说明的要求”进行 TCB 的开发。

经过多年攻关努力，中国科学院软件研究所、国防科技大学等国内少数几家研究机构开发了满足国标第四级要求的国产安全操作系统原型，例如银河麒麟、中科安盛 v4.0 等，但是目前国内尚未有符合访问验证保护级要求的安全操作系统成果。在高等级安全操作系统研发中，形式化保证技术一直是公认的主要难点之一。

近年来随着虚拟机、微内核等新型系统架构的出现，操作系统出现层次化、轻量化的特点，为操作系统的验证分析打下了良好的基础。在“面向访问验证保护级的安全操作系统原型系统研发”课题的研究中，我们提出了基于虚拟监控机的高安全等级操作系统架。整个系统由计算机硬件平台、基于 TCM（Trusted Cryptography Module）的可信 Bootloader、安全虚拟监管引擎 CASVisor 和虚拟域 OS 4 个层次组成。其中，CASVisor 是在 Xen 基础上裁剪得到的一个轻量 VMM，初步估算由 1.5 万行 C 代码实现，主要包含调度、虚拟域管理、域间通信、内存管理等模块。此外，CASVisor 还包括强制访问控制模块（MAC），用于实施域间隔离和访问控制。位于 CASVisor 上的 VM（Virtual Machine）是各种实际系统的运行域。整个系统以 CASVisor 为基础，具有强隔离、层次化、模块化的特点。

CASVisor 采用了以函数式建模技术和定理证明技术为主的形式化研发方案，如图 1 所示。图 1 主要包括 3 个方面的工作：以 Haskell 语言为主为系统建立形式化模型；以 Isabelle/HOL 为主形成系统的形式化规范，并分析系统安全模型和形式化规范之间的对应性；基于系统的快速原型和规范用于指导 C 代码的实现，并分析实现机制的部分正确性。本文主要介绍第一部分的工作，用函数式编程语言 Haskell 来构建一个可执行的带安全策略的 VMM 快速原型，工作主要包括：

1）X86 架构机器模拟器设计和实现；

2）CASVisor 原型系统的设计和实现；

3）简单的 VM 系统，调用 CASVisor 的系统接口，完成初始化及信息输出的功能。

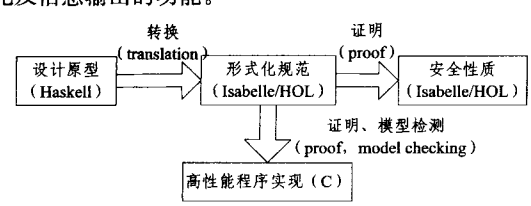

图 1 CASVisor 的形式化研发方案

本文第 2 节绍与形式化安全操作系统相关的工作；第 3 节介绍 CASVisor 的系统架构；第 4 节介绍 CASVisor 系统的内核模型；第 5 节介绍 CASVisor 系统的 X86 模拟器模型；第 6 节展示运行 CASVisor 系统的结果；最后进行总结。

2 相关工作

国外早期高等级安全操作系统的研发过程中都采用了各种形式规范和验证技术。LOCK、ASOS 和 DTOS 被公认为是形式化验证方法在操作系统开发中应用得较为成功的几个项目，前两者使用 Gypsy 验证环境（采用 Gypsy 规范语言），后者使用 Z 语言作为规范语言[3]。

近年来开展的操作系统验证分析的工作中，很多都以微小内核为研究对象。例如，澳大利亚 NICTA L4. verified 小组的工作以 EAL7 为目标，验证分析安全微内核 seL4[4]，他们提出了一种操作系统开发的高可信形式化分析方法，主要内容包括基于 Haskell 实现的系统快速原型和定义系统的功能接口。Haskell 的系统原型可以转化为 Isabelle/HOL 的形式化规范，在 Isabelle/HOL 的验证环境中对系统规范逐步求精。最后，基于 C 语言验证环境分析 Isabelle/HOL 规范并实现 C 代码的对应性。L4. verified 项目的工作得到了国际上的认可，近期他们已经完成约 8,0000 行 C 代码的验证分析工作，实现了“bug free”的 seL4 内核原型。

德国的 Verisoft 项目试图将形式化验证技术应用到编译器、操作系统以及大规模集成电路的验证分析中。在其最近发布的阶段性成果中，提出了一个微小内核 VAMOS 和其上 C 语言子集编译器的形式化规范，并验证了从精简的 email 服务程序到 C 语言语义的对应性[5,11]。

美国海军研究所正在进行的 Xenon 是一个形式化研发项目，其目标是在开源 Xen 系统的基础上研发面向 EAL5 的高可信虚拟机监控器。该课题组近期提出了基于 CSP 的安全控制模型和验证方法，设计了基于 Z 和 Circus 的验证方法，并分析了 Xen 中 MMU 模块的半形式化规范，提出了改进方法[6,12]。

此外，还有很多知名机构也在进行操作系统验证分析的工作，例如耶鲁大学 FLINT 小组、荷兰 Nijmegen 大学等在对 Nova 虚拟机进行分析验证。

3 系统架构

CASVisor 参考了 Xen 架构，如图 2 所示。它包含域（Domain）空间、内核模型和模拟器 3 个部分。其中域空间包含 Dom0 和多个 DomU；内核模型参考了 Xen 的内核模型构架，主要包括 Domain 管理、VCPU（虚拟 CPU）、事件通道、授权表、调度机制以及 MAC 模块；X86 模拟器模拟物理计算机的机器指令、物理内存、中断以及与硬件相关的其他部分，为 CASVisor 系统的运行提供一个虚拟的硬件环境。

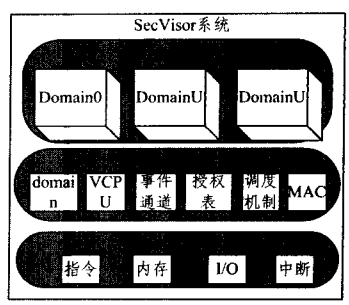

图 2 CASVisor 系统架构

CASVisor 的设计采用内核变迁思想，将系统的状态表示为内核状态 Monad，每条指令的返回类型都为内核状态，模拟器每执行一条指令都将内核状态从一个状态变迁为另一个状态。在 CASVisor 的模拟过程中，整个系统作为一个用户进程运行，模拟器驱动并管理整个进程的运行过程。各 Dom 需要执行的操作最终通过模拟器中定义的指令完成。模拟器和内核模型的接口主要包括硬件中断、系统调用、保存/恢复、指令以及虚拟和物理内存几个部分，如图 3 所示。

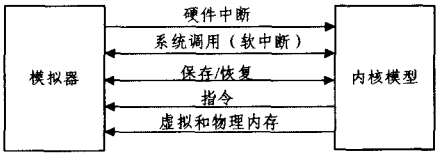

图 3　模拟器与内核模型之间的接口

4　SecVior 系统的内核模型

内核模型是 Xen 架构的 VMM 层，主要包括 Domain、调度机制、Grant-table、Event-channel、MAC 模块几个部分。它将模拟器提供的物理 CPU 进一步虚拟化为多个 VCPU，以分配给 Domain0 以及 DomainU。每个 Domain 都可以拥有一个或者多个 VCPU，并提供 Domain 之间的通信机制。

4.1　Domain

Domain 主要存储操作系统运行所需的数据信息。它的主要信息包括其标识 domid、强制访问控制相关的引用标识 ssidref、拥有的 vcpu 列表、代码段、数据段和堆栈段，以及共享内存指针和事件通道列表。Domain 数据结构的定义如下所示：

```
data Domain =Domain
{
  domid::DomID,--Domain 标识
  ssidRef::SsidRef,--MAC 标识
  vcpu ::[Int],--VCPU 列表
  codespace ::Int,--代码段
  dataspace ::Int,--数据段
  stackspace::Int,--堆栈段
  eventchannel::[EventChannel]--事件通道
  ... ...
}
```

Domain 是系统中运行的主体，其主要接口包括 Domain 的创建、查询、删除等操作。鉴于我们关注的中国墙策略主要涉及 Domain 的创建操作，目前提供的 Domian 的操作接口函数如下所示：

```
createDomain::(DomID,[Int],Int,Int,Int,Int)-> Instruction
getCurrentDomain::Machine-> Domain
getDomainByID::Machine-> DomID-> Domain
```

createDomain 函数用于创建一个新的 Domain，其过程包含为该 Domain 创建新的数据信息、初始化新创建的 Domain 的共享内存信息以及初始化事件通道等。getCurrentDomain 函数从当前机器状态获取当前正在运行的 Domain。getDomainByID 函数通过 domid 获取指定的 Domain。

4.2　调度机制

虚拟化技术中 CPU 的调度是关键，它决定哪个 VCPU 能够真正获得在底层上的物理 CPU。在 Xen 中早期集成了循环调度(Round Robin Scheduling)算法。为了提高效率，在最新版本中提供了两种调度机制：Credit 算法和简单最早期限优先(SEDF)算法。为了简化 SecVior 原型系统，内核模型中我们采用了循环调度算法，将物理 CPU 时间片循环分配给各个运行的 Domain。在 CASVisor 中，调度信息由数据结构 ScheduleData 来定义，它保存了所有 VCPU、Domain 以及当前正在运行的 VCPU 信息，如下所示：

```
data ScheduleData=ScheduleData
  {
    vcpus::[VCPU],--所有 VCPU
    domains::[Domain],--所有 Domain
    current_vcpu::Int --正运行的 VCPU
}
```

每当 Domain 的 CPU 用完分配的时间片后，调度函数会用循环调度算法选择下一个被调度的 VCPU，然后切换上下文运行该 VCPU。调度算法的过程如下所示：

```
roundRobinScheduler::Instruction
roundRobinScheduler=do
  m <-get
  let(current,v)=currentVCPU m
  let s=slices v
  if s == 0
    then do
      --选择新的 VCPU
      let next=selectNext(vcpus $ getScheduleData m) current
      --分配时间片
      setVCPUSlices current fullSlices
      --切换上下文
      contextSwitch current next
          else do
              setVCPUSlices current(s-1)
```

4.3　Grant-table

授权表是实现 Domain 之间内存共享的主要机制，它不但可以实现将一个 Domain 的共享内存区域授权给其他 Domain 访问，也能实现向授权的其他 Domain 以传递内存的方式发送数据，即可以实现内存映射和内存传递。

Domain 中的每条授权信息都被定义为授权表中的一项，即授权项，授权表就是授权项的列表。授权项的信息包括授权目标 Domain、授权的虚拟内存地址以及与该授权项相关的标志(例如允许内存传递等)。授权项的数据结构定义如下所示：

```
data GrantEntry=GrantEntry
{
  ge_flags::[GrantTableFlag],--授权标志
  ge_domid::Int,--提供 Domain 的 id
  ge_frame::VPtr --授权的虚拟内存地址
}
```

CASVisor 提供了一系列关于授权表的 Hypercall，以方便共享内存的操作。目前支持的操作包含建立授权表与建立授权引用，如下所示。

```
gnttabSetupTable::Int --建立授权表
gnttabSetupTable=2
gnttabMapGrantRef::Int --建立授权引用
gnttabMapGrantRef=0
```

4.4 Event-channel

事件通道是 Domain 与 Domain 之间以及 Domain 与 CASVisor 之间的异步通信机制。在它们通信之前，首先需要在两端建立通信的事件通道(event channel)。事件通道的数据结构如以下列表所示，它包含事件通道当前所处的状态(state)、是否为 xen 的标识(custom_is_xen)、关联的 VCPU(notify_vcpu_id)以及与该事件通道绑定的另一端的 Domain 的 domid 等标识。

```
data EventChannel=EventChannel
{
  State::EventChannelState,
  consumer_is_xen::Bool,
  notify_vcpu_id::Int,
  unbound_remote_domid::DomID,
  interdomain_remote_dom::DomID,
  interdomain_remote_port::Int,
  pirq::Int,
  virq::Int
}
```

在一个 Domain 希望与另一个 Domain 通信时，需要在这两个 Domain 间创建一个事件通道。目前 CASVisor 支持的事件通道操作包括通道的初始化、发送事件、事件通道的绑定以及 Domain 间的绑定。以下列表为 CASVisor 提供与事件通道相关的 Hypercall 功能号：

```
evtchnAllocUnbound= 6::Int
evtchnBindInterdomain= 0::Int
evtchnClose= 3::Int
evtchnSend= 4::Int
```

4.5 MAC 模块

中国墙(Chinese wall，简称 CW)策略的最初构想来源于证券的交易[17]，如今有很多论文给出其形式化模型，以保证 CW 策略的正确实现[7-9]。CW 策略已经用于各种各样信息流的控制，如 Xen 虚拟化技术已经采用该策略来控制运行不同工作任务的 Domain 同时运行。

CW 策略可以阻止有冲突的 Domain 同时运行，从而保证 Domain 之间通信的安全性，防止 Domain 间的隐蔽信道。而 STE 策略是一种对 Domain 之间通信的细粒度控制机制。CW 策略有两个性质：CW-简单访问规则和 CW-* 特性规则。CW-* 特性规则通过记录 Domain 启动的历史记录，有效防止虚拟机之间通信的隐蔽信道问题。CASVisor 采用一种改进的 CW 策略[10]和 STE 策略来实现 Domain 之间的隔离以及通信的安全性。PCW 策略可以削弱 CW-* 属性限制造成的虚拟机并行度下降的问题。本文将主要介绍 PCW 策略的设计与实现。

以下列表是 PCW 策略的核心数据结构，其中 conflictAggregateSet 数组和 accessed 数组是 PCW 策略数据结构中最重要的两个成员，分别表达 PCW 策略的简单属性和弱化的 * 属性规则。

```
data ChwallBinaryPolicy=ChwallBinaryPolicy {
... ...
    ssidrefs::TDArray,--标签对应的 CW 类型
    conflictAggregateSet::[Int],--冲突类型
    runningTypes::[Int],--运行的 CW 类型
    conflictSets::TDArray--冲突集
    accessed::TDArray --与历史相关冲突类型}
```

PCW 策略通过在源码中添加安全钩子函数的方式来控制 Domain 的创建以及删除。在创建虚拟机时除了判断该虚拟机与正在运行的虚拟机是否相冲突外，还应该判断它是否满足弱化的 * -属性规则。

```
chwallDomainCreate::ChwallBinaryPolicy->Domain->(Chwall-
BinaryPolicy,Bool)
chwallDomainDestroy::ChwallBinaryPolicy-> AcmSsidDomain->
Domain-> ChwallBinaryPolicy
```

5 X86 模拟器模型

X86 模拟器为 CASVisor 的运行提供一个虚拟的物理机器，以屏蔽一些硬件细节，降低 CASVisor 系统的实现复杂度。

5.1 机器结构

机器模拟器由 C 和 Haskell 语言混合实现，如图 4 所示。

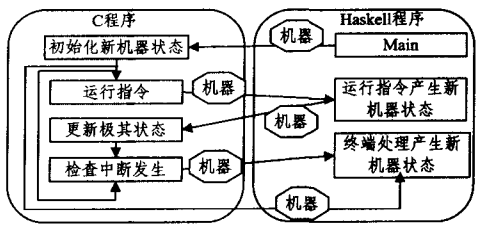

图 4 CASVisor 系统模拟器设计框图

C 语言主要负责中断的实现以及程序的循环运行，以搭起模拟器的运行框架；而 Haskell 程序部分主要负责机器状态的维护和更新。CASVisor 将机器状态封装为 Haskell 的 Monad，并用 Monad 的操作表达机器状态的迁移，从而获得具有严格语义定义的内核状态模型及其迁移关系。机器模型数据结构如下：

```
data Machine=Machine
{
  processor::CPU,--物理 CPU
  memory::(          --虚拟内存
       CodeSegments,--代码段
       DataSegments,--数据段
       StackSegments --堆栈段
  ),
  physmem ::PSpace,--物理内存
  intvec::[(Int,Int)] --中断向量表
}
type MachineState=StateT Machine IO()
```

机器指令的语义可以描述为机器状态改变，程序则是指令列表。用户级程序的执行也就是执行一系列的指令，完成机器状态的迁移，如下所示：

```
type Instruction =MachineState
type Program=[Instruction]
```

5.2 指令部分

对于模拟器的指令集，CASVisor 目前主要实现了 X86 指令集中的一些关键的指令，称作元指令。主要包括 mov 指令，将整数值传递给寄存器；out 指令，向指定终端输出字符串；jmp 用于段内跳转，以隐式地改变 CPU 的 EIP；而 ljmp 指令为段间的跳转(此时需要改变 CPU 的 CS 与 EIP 寄存器)；int 指令用于陷入内核执行软中断。其他的指令还包括 halt，iret，push 和 pop 等指令。以下是 CASVisor 的关键指令列表。

```
mov::Register—> Int—> Instruction
out::Int—> String—> Instruction
jmp::Int—> Instruction
ljmp::Int—> Int—> Instruction
int::Int—> Instruction
halt::Instruction
iret::Instruction
push::Int—> Instruction
pop::Register—> Instruction
…… ……
```

此外，在程序中，直接使用汇编级的元指令编程，效率会较低，因此可以将多个元指令和一些高级的 Haskell 函数封装成一条指令，这些指令无论功能如何，均为对机器状态的改变。如

```
XXXX::Instruction
XXXX=do ……
```

封装的指令虽然会增加指令粒度，但不会影响要验证的 CASVisor 模型，而且可以简化程序的编写。

5.3 内存部分

CASVisor 采用了和 SeL4 相同的内存模型，内存分为物理内存和虚拟内存。物理内存的类型为 PSpace，它是一个 Dynamic 类型的二叉树，其定义如下：

type PSpace=BinaryTree Dynamic

在这个简化的模型中，Pspace 只是维护了物理地址到内核对象的一个映射，并非传统意义上的物理内存。存储在物理地址空间中的对象是用来动态分配以支持内核对用户级的程序的抽象。存储在本文设计的物理地址空间中的数据类型包括调度信息数据(ScheduleData)、陷入内核数据(TrapData)、授权表信息(GrantTable)以及安全策略相关信息(AcmData)等。

物理内存操作包括读、写等操作，如以下列表所示。其中，getPhysObject 从给定地址的物理地址空间的内存中返回存储对象；setPhysObject 将给定地址的内容设为新的对象，并返回修改后的新物理地址空间。

```
getPhysObject::Pspace—> PPtr—> Dynamic
setPhysObject::Pspace—> PPtr—> Dynamic—> Pspace
```

模拟器还提供了虚拟内存机制，以模拟真实机器中的 MMU。虚拟内存提供了虚拟地址向物理内存地址的映射。在 CASVisor 系统的模拟器中，省略了传统 X86 的页表、段描述符等机制，仅仅做了一个地址到另一地址的简单映射，但能实现虚拟地址和物理地址的多对多的任意映射。CASVisor 中的授权表、各 Domain 内部独立的内存管理均通过虚拟内存得以实现。

如图 5 所示，每个 Domain 都有各自的虚拟地址空间 Vspace，在默认情况下每个 Domain 的虚拟地址空间被映射到物理地址空间中的一部分，但是通过修改“页表”(在模拟器中为映射项)，不同 Domain 中的虚拟地址可以映射到同一物理地址。

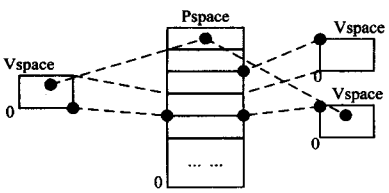

图 5 CASVisor 系统虚拟地址与物理地址的映射关系

对虚拟内存的操作有 setObject，getObject 等，如以下列表所示。其中 setObject 函数将 Dynamic 对象存储到虚拟地址 Vptr 对应的物理地址中，并返回新的机器状态；getObject 函数从虚拟地址 Vptr 对应的物理地址中获取一个 Dynamic 对象。

```
setObject::Machine—> VPtr—> Dynamic—> Machine
getObjectV::Machine—> Int—> VPtr—> Dynamic
```

5.4 中断部分

CASVisor 的模拟器也参考 X86 中断机制的设计并实现了中断机制，它目前实现了时钟中断以及 int 等软中断。在 X86 模拟器的机器结构中为了简化设计，将中断向量表 intvec 直接放入数据结构 Machine 中。intvec 是由(cs，eip)元组列表形成中断向量表，它可以根据中断号在列表中选择中断处理程序地址，以模拟真实的向量表结构和中断描述符等，完成传统意义上的中断向量表的功能，实现该模拟器的中断机制。以下列表是一部分中断号：

```
intrZero= 0 ::Int
intrTrace= 1::Int
intrBreakpoint= 2::Int
intrPageFault= 3::Int
intrTimer= 4::Int
intrHypercall= 6::Int
```

6 运行 CASVisor 系统

目前 CASVisor 在应用级支持一个 Dom0、两个 DomU (Dom1 与 Dom2)和一个空闲 Domain(IdleDom)。Dom0 创建 Dom1 和 Dom2 后开始循环执行空操作。Dom1 的主要操作是创建授权表并绑定到 Dom2 的事件通道，然后将一个字符串共享给 Dom2。Dom2 可以执行事件通道绑定操作，并获取 Dom1 共享的字符串。

CASVisor 系统提供了一个模拟的终端来显示系统内核以及各个 Domain 在运行过程中的状态，内核信息以及各个 Domain 的状态可以显示在终端的窗口中。如图 6 所示，内核信息显示到第一个终端(左上)，Domain0 的状态信息显示到第二个终端(右上)，Domain1 和 Domain2 的状态信息依次显示到第三个(左下)和第四个(右下)终端上。

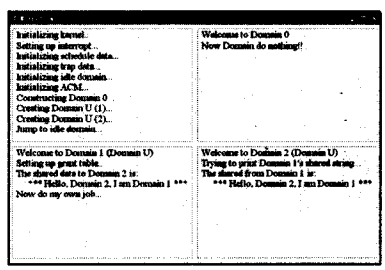

图 6 未加载强制访问控制模块的 CASVisor 系统运行结果

图 6 是 CASVisor 未加载强制访问控制模块时的运行结果。第一个终端上显示内核调用信息，包括内核初始化、设置中断向量表、初始化内核调度数据、ACM 访问控制数据等，并显示成功创建了 Domain0，IdleDomain，Domain1 以及 Domain2。在第二、三、四个终端上分别显示 Domain0，Domain1，Domian2 的动作信息，从第三、四个终端上可以看到 Domain1 将字符串“* * * Hello，Domain 2，I am Domain 1 * * *”成功共享给 Domain2。

如果将 Domain1 和 Domain2 处于同一个冲突集中，按照中国墙策略的性质，创建 Domain1 后再创建 Domain2 会失败，此时会得到如图 7 所示的调试终端错误信息。从图 7 可知，Domin2 拥有第 2 种中国墙类型，正在运行的中国墙类型是第 0、1 两种，其存在一个包含第 1、2 两种中国墙类型的冲突集，而与正在运行的中国墙类型冲突的中国墙类型是第 2 种，因此 Domain2 会因它拥有第 2 种中国墙类型而被中国墙策略拒绝创建。

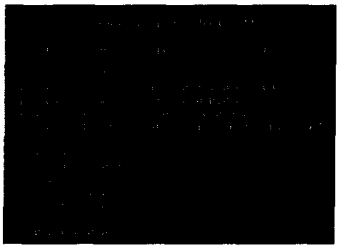

图 7　中国墙策略简单属性验证

图 8 是在创建 Domain1 后、创建 Domain2 之前关闭 Domain1 的情况下得到的调试终端错误信息。从图 8 可知，现在正在运行的中国墙类型为第 0 种，而且不存在与正在运行的中国墙类型冲突的类型。但是此时的 accessed 数组表明正在运行的第 0 种中国墙类型与第 2 种类型冲突，而 Domain2 正好拥有第 2 种类型，因此中国墙策略的弱化 * 属性规则拒绝 Domain2 的创建。

图 8　中国墙策略弱化 * 属性规则验证

图 6、图 7 以及图 8 表明 CASVisor 不但能正确地模拟 HyperVisor 的功能，而且能正确地实现强制访问控制，为 Domain 的运行提供了一个安全的隔离环境。

结束语　本文描述了一个面向“访问验证保护级”的安全操作系统的形式化原型和顶层规范，它满足 TCSEC 以及 GB17859-1999 中的有关要求：“FTLS 必须包括系统功能模块、硬件、固件等的形式化抽象定义”。我们的方法是用函数语言 Haskell 严格刻画 TCB、硬件、固件的形式化语义。此外，系统还具备模拟功能，支持快速原型的开发方法，可用于指导 C 程序的设计和实现。

现有的工作是中国科学院知识创新工程重要方向项目“面向访问验证级操作系统原型研发”的一部分。将来的工作主要是在该系统原型的基础之上进行形式化的验证分析工作，主要包括将 CASVisor 模型转化为 Isabelle/HOL 规范，分析系统模型对 CW 性质的满足性；构建 CW 策略的形式化模型（已基本完成），基于数据求精的思想分析系统规范和 CW 策略模型之间的对应性。

参 考 文 献

[1] Klein G. Operating system verification—an overview[J]. Sadhana, 2009, 34(1): 27-69

[2] U. S. Department of Defense. Trusted Computer System Evaluation Criteria[M]. DoD 5200. 28-STD. 1985

[3] Secure Computing Corp. DTOS Formal Security Policy Model [R]. DTOS CDRL A004. 2675 Long Lake Rd, Roseville, MN 55113, Sept. 1996

[4] Derrin P, Elphinstone K, Klein G, et al. Running the Manual: An Approach to High-Assurance Microkernel Development[C]// Proceedings of the 2006 ACM SIGPLAN Workshop on Haskell. Portland, Oregon, 2006: 60-71

[5] Hillebrand M A, Paul W J. On the architecture of system verification environments[J]. Lecture notes in computer science, 2008, 4899: 153-168

[6] McDermott J, Freitas L. A formal security policy for xenon[C]// Conference on Computer and Communications Security, Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering. 2008: 43-52

[7] David Dr, Brewer F C, Michael Dr, et al. The Chinese Wall Secutity Policy[C]// IEEE Symposium on Security and Privacy. 1989: 206-214

[8] Lin T Y. Chinese wall security policy-an aggressive model[C]// Fifth Annual Computer Security Applications Conference. 1989: 282-289

[9] Dallons G, Massonet P, Molderez J-F, et al. An Analysis of the Chinese Wall Pattern for Guaranteeing Confidentiality in Grid-based Virtual Organisations[C]// Third International Conference on Security and Privacy in Communications Networks and the Workshops. 2007: 217-222

[10] Cheng Ge, Jin Hai, Zou Deqing, et al. A Prioritized Chinese Wall Model for Managing the Covert Information Flows in Virtual Machine System[C]// The 9th International Conference for Young Computer Scientists. 2008: 1481-1487

[11] Inder R T, Alexandra T. CVM—a verified framework for microkernel programmers[C]// Proceedings of the 3rd International Workshop on Systems Software Verification (SSV08)'. ElectronicNotes in Computer Science, 2008, 217: 151-168

[12] James K Jr, McDermott J, Kang Myong, et al. Documenting Xenon's Page_Alloc Module[R]. CHACS, 2007: 40

[13] 卿斯汉，刘文清，温红子，等. 操作系统安全[M]. 北京：清华大学出版社，2004

[14] 国家质量技术监督局. 计算机信息系统安全保护等级划分准则[S]. GB17859-1999

[15] 国家质量监督检验检疫总局. 信息安全技术信息系统通用安全技术要求[S]. GB/T20271-2006

[16] 国家质量监督检验检疫总局. 信息安全技术操作系统安全技术要求[S]. GB/T20272-2006

[17] 秦超，陈钟，段云所. Chinese Wall 策略及其在多级安全环境中的扩展[J]. 北京大学学报，2002, 38(3)