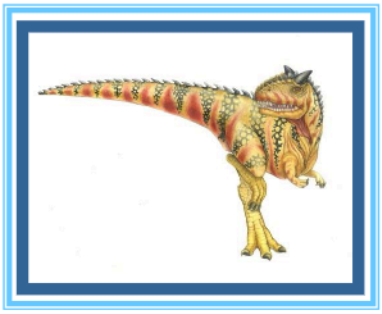
第 17 章：保护



第 17 章：保护

- 保护目标 保护环原则 保护环 保护域 访问矩阵
- Access Matrix 的实现

目标

- 讨论现代计算机系统中保护的目标和原则 说明如何使用保护域与访问矩阵相结合来指定进程可以访问的资源
- 检查基于能力的保护系统

保护目标

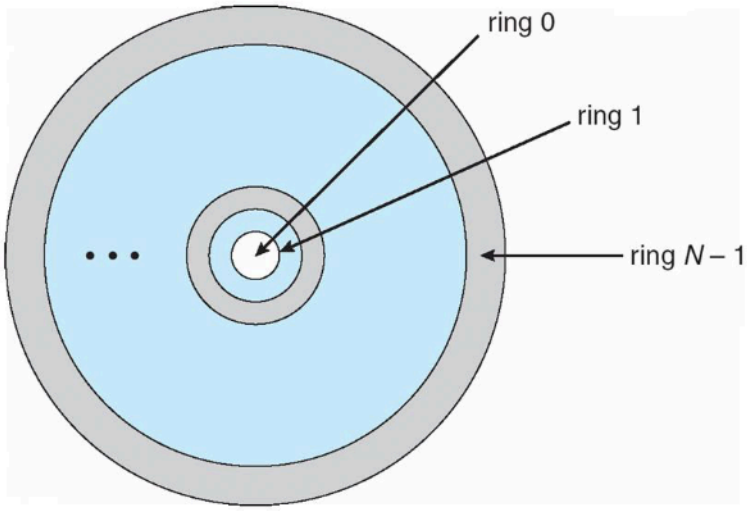
- 在保护模型中，计算机系统由对象、硬件或软件的集合组成
  - 硬件对象：CPU、内存段、打印机、磁盘和磁带软件对象：文件、程序和信号量
  -
- 每个对象都有一个唯一的名称，可以通过一组定义明确的操作进行访问
- 保护问题是确保每个对象都被正确访问，并且只能由允许这样做的进程访问
- 机制与策略不同，在策略中，机制决定如何执行某事，而策略则决定将要执行哪些操作。
  - 这种分离对于灵活性很重要，因为政策可能会因地或时而异。
  - 这种分离确保了并非每次政策变化都需要改变底层机制。

保护原则

- 指导原则 – 最小权限原则
  - 程序、用户和系统应该被授予足够的权限来执行他们的任务 - 缓解攻击
  - 在文件权限中，此原则规定用户具有读取访问权限
    但不能写入或执行对文件的访问权限。最小权限原则要求 OS 提供一种机制，仅允许读取访问，而不允许写入或执行访问
- 正确设置权限（即对对象的访问权限）可以限制实体出现错误或被滥用时的损害

保护环

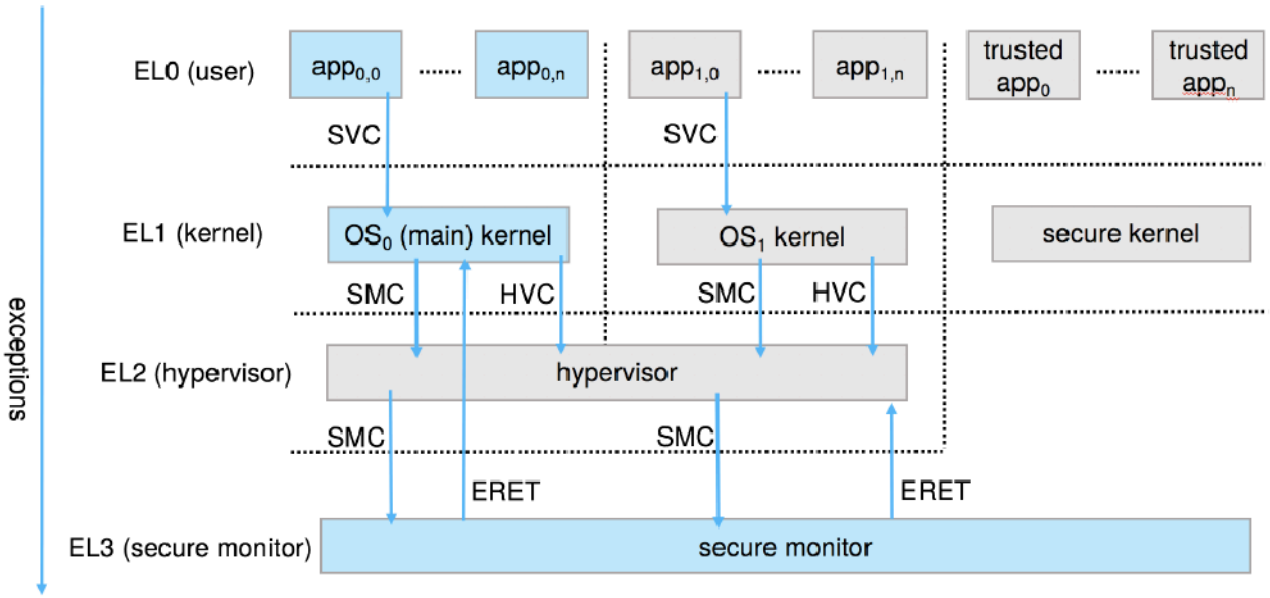
- 用户模式和内核模式 – 权限分离支持单独执行概念所需的硬件支持
-
- 设 Di 和 Dj 为任意两个域环 If j < i □ Di □ Dj
-
- 最内层的环 （环 0） 提供完整的权限集



保护环 （续）

- 按权限数量排序并相互保护的组件 例如，内核位于一个环中，而用户应用程序位于另一个环中 这种权限分离需要硬件支持
  - 用于在环之间传输的 “Gates” ，例如 syscall Intel 指令，也用于 trap 和 interrupt
- 引入了虚拟机管理程序 （Intel） （另一个环） - 虚拟机管理器，用于创建和运行虚拟机，并且具有比来宾操作系统的内核更多的功能
- ARM 处理器添加了 TrustZone 或 TZ 环，以保护具有访问权限的加密功能（比内核特权更高）
  - 这个最高特权的执行环境拥有对硬件支持的加密功能的独占访问权限，例如 NFC 安全元件和片上加密密钥，从而使处理密码和敏感信息更加安全。

ARM CPU 架构


EL0 (user)
app0,0
app0,n
app1,0
app1,n
trusted app0
trusted appn
SVC
SVC
EL1 (kernel)
OS0 (main) kernel
OS1 kernel
secure kernel
SMC
HVC
SMC
HVC
EL2 (hypervisor)
hypervisor
SMC
SMC
ERET
ERET
EL3 (secure monitor)
secure monitor
exceptions

保护域

- 保护环将功能分为不同的域，并按层次结构对它们进行排序
- 域可以被认为是没有层次结构的环的泛化 计算机系统可以被视为进程和对象
  - 硬件对象（如 CPU、内存、磁盘）和软件对象（如文件、程序、信号量）
- 例如，流程应该只能访问它当前需要完成其任务的对象 - 需要知道原则（策略）
- 可以通过在保护域中运行的进程来实现
  - 保护域 指定进程可以访问的资源集 每个域指定对象集和可对每个对象调用的操作类型

域保护 （续）

- 对对象执行操作的能力是一种访问权限 域是访问权限的集合，每个访问权限都是一对有序的 <object-name， rights-set>
  - 例如：如果域 D 具有<文件 F{read，write}> 的访问权限，则在域 D 中执行的进程可以读取和写入文件 F。但是，它不能对该对象执行任何其他操作。
- 域可以共享访问权限如果进程可用的资源集在进程的整个生命周期中是固定的，则进程和域之间的关联可以是静态的，也可以是动态的
- 如果关联是动态的，则可以使用一种机制来允许域切换，从而使流程能够在执行的不同阶段从一个域切换到另一个域

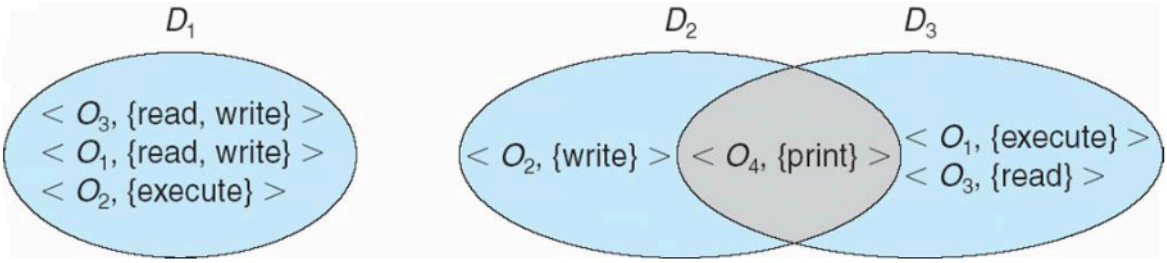
域保护 （续）

Domain 可以通过多种方式实现：

- 每个用户都可能是一个域 - 可以访问的对象集取决于用户的身份。更改用户时发生域切换
- 每个进程都可能是一个域 - 可以访问的对象集取决于进程的身份。当一个进程向另一个进程发送消息，然后等待响应时，会发生域切换。
- 每个过程都可能是一个域 - 可访问的对象集对应于过程中定义的局部变量。进行过程调用时发生域切换

域结构

- Access-right = <object-name， rights-set>其中 rights-set 是可对对象执行的所有有效操作的子集
- 域 = 访问权限集
- 域 D2 和 D3 共享访问权限 <O4， {print}>因此，在这两个域中执行的进程都可以打印对象 O4。



访问矩阵

- 以矩阵形式查看保护（访问矩阵） 行表示域，列表示对象
- Access（i，j） 由一组访问权限组成 - 在 Domaini 中执行的进程可以在 Objectj 上调用的一组操作

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix 的使用

- 访问矩阵方案提供了指定各种策略的机制 - 机制和策略分离
- 该机制包括实现访问矩阵和确保语义属性成立。
  - 确保在域 Di 中执行的进程只能访问行 i 中指定的对象。
- 策略决策指定第 （i，j） 个条目中应包含哪些权限，并确定执行每个进程的域
- 如果域 Di 中的进程尝试对对象 Oj 执行 “op”，则 “op” 必须位于访问矩阵中
- 创建对象的用户可以定义该对象的访问列
  - 当用户创建新对象 Oj 时，列 Oj 将添加到访问矩阵中，并按照创建者的规定使用相应的初始化条目。用户可以根据需要决定在列 j 的某些条目中输入一些权限，并在其他条目中输入其他权限。

使用 Access Matrix （Cont.）

这可以扩展到动态保护

- 添加、删除访问权限的操作特殊访问权限：
  - Oi 的所有者 - 可以在 copy op 列中的任何条目中添加和删除从 Oi 到 Oj 的任何权限（用“*”表示）- 仅在列内（即，对于对象）
  - control – Di 可以修改 Dj 访问权限 – 修改域对象（一行）
  - transfer – 从域 Di 切换到 Dj
- 适用于对象的 Copy 和 Owner - 更改列中的条目 Control applied to domain object - 更改行中的条目 可以动态创建新对象和新域并将其包含在访问矩阵模型中
- 在动态保护系统中，我们有时可能需要撤销对不同用户共享的对象的访问权限 – 撤销访问权限

图 A 的访问矩阵，其中 Domains 作为对象

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

具有版权的访问矩阵

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

具有所有者权限的访问矩阵

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

图 B 的修改访问矩阵

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Access Matrix 的实现

- 通常，访问矩阵是稀疏的;也就是说，大多数条目将是空的
- 选项 1 – 表 A 中的全局表格存储有序三元组 <domain, object， rights-set> 请求对域 Di 中的对象 Oj 执行操作 M> 搜索表中的 < Di、Oj、Rk > 和 M ∈ Rk
  - 但是表可能很大 > 可能无法放入主内存，需要额外的 I/O - 经常使用虚拟内存技术
  - 难以对对象进行分组 - 例如，如果每个人都可以读取特定对象，则此对象必须在每个域中具有单独的条目。

Access Matrix （Cont.） 的实现

- 每列 = 一个对象的访问控制列表 定义谁可以执行什么操作

域 1 = 读取，写入域 2 =
读取域 3 = 读取

- 每行 = Capability List （like a key） 对于每个域，允许对哪些对象执行哪些操作

对象 F1 – 读取 对象 F4 – 读取、写入、执行 对象 F5 – 读取、写入、删除、复制

Access Matrix （Cont.） 的实现

- 选项 2 – 对象的访问列表 作为一个对象的访问列表实现的每个列 生成的每个对象列表由有序对<域、权限集>组成，定义具有对象非空访问权限集的所有域
  - 显然，空条目可以被丢弃。这可以很容易地扩展以定义默认的访问权限集 ->如果 M ∈默认设置，则还允许访问（适用于所有域）

Access Matrix （Cont.） 的实现

- 选项 3 – 域的功能列表 域的功能列表不是基于对象的，而是基于域的 域的功能列表 域的功能列表是对象列表以及允许对它们执行的操作
  - 由其名称或地址表示的对象，称为能力 要在对象 Oj 上执行操作 M，进程请求操作 M，将对象 Oj 的能力（或指针）指定为参数 拥有能力意味着允许访问
  -
- 与域关联的功能列表，但从不由在该域中执行的进程直接访问
  - 相反，功能列表本身是一个受保护的对象，由 OS 维护，并且仅由用户间接访问
  - 这避免了用户修改功能列表的可能性 如果所有功能都是安全的，则他们保护的对象也是安全的，可以防止未经授权的访问
  -

Access Matrix （Cont.） 的实现

- 选项 4 – 访问列表和功能列表之间的锁键妥协 每个对象都有唯一的位模式列表，称为锁 每个域作为称为键的唯一位模式列表 如果域具有与对象的一个锁匹配的键，则域中的进程只能访问对象
  - 与功能列表一样，域的键列表必须由操作系统代表域进行管理。
  - 不允许用户直接检查或修改键（或锁）列表。

实施比较

选择实现访问矩阵的技术涉及各种权衡。

- 全局表简单但很大，缺少对象或域的分组访问列表直接与用户的需求相对应
- 在用户创建对象时指定对象的访问列表确定每个域的访问权限集很困难 - 必须检查对对象的每次访问，需要搜索访问列表。
- 用于本地化给定流程信息的功能列表
- 但是吊销功能可能效率低下 锁键可以有效且灵活，具体取决于密钥的长度 密钥可以在域之间自由传递，轻松吊销
- 大多数系统使用访问列表和功能的组合

第十七章结束