OPERATING SYSTEMS

Protection

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

- 目录
  - 概述
  - 保护原则
  - 保护域
  - 访问矩阵
  - 访问矩阵的实现
  - 访问控制
  - 访问权限的撤回
  - 基于能力的系统
  - 基于语言的保护

概述

- 在一种保护模型中，计算机由一组对象、硬件或软件组成
- 每个对象都有一个唯一的名称，可以通过一组定义良好的操作进行访问
- 保护问题
  - 确保每个对象都被正确访问，并且只能由允许访问的进程访问

保护原则

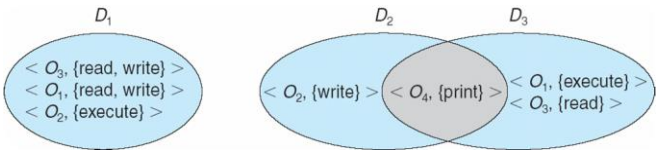
- 指导原则——最低特权原则
  - 程序、用户和系统应只被授予执行其任务所需的最低权限
- 限制实体有缺陷、被滥用时的损害
- 可以是静态的，也可以是动态的：
  - 静态的
    - 在系统生命期间，在进程生命期间
  - 动态（根据需要由进程更改）
    - 域切换、权限提升
- “需要才知道(need to know)”是关于数据访问的类似概念

保护原则

- 必须考虑“粒度”方面
  - 粗粒度
    - 权限管理更容易、更简单，但现在基于大块实现最低特权原则
    - 示例 — 传统Unix进程要么具有关联用户的能力，要么具有root用户的能力
  - 细粒度
    - 管理更复杂，开销更大，但保护性更强
    - 示例 — 文件ACL访问控制列表，RBAC基于角色的访问控制

域结构

- 访问权限 = <对象名，权限集>
  其中权限集是可以对对象执行的所有有效操作的子集
- 域 = 访问权限集合

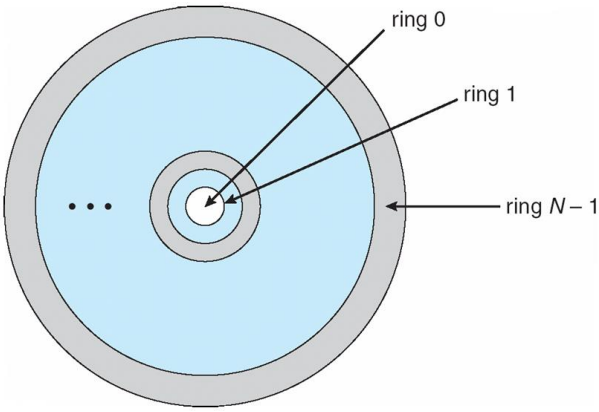


域的实现(UNIX)

- 域 = 用户id
- 通过文件系统实现域切换
  - 每个文件都与一个域位（setuid位）关联
  - 当文件被执行且 setuid = on 时，用户id被设置为正在执行的文件的所有者
  - 执行完成后，将重置用户id
- 通过密码实现域切换
  - 当提供其他域的密码时，su命令临时切换到其他用户的域
- 通过命令进行域切换
  - sudo命令前缀在另一个域中执行指定的命令（如果原始域具有权限或使用密码）

域的实现(MULTICS)

- 设 D_i 和 D_j 是任意两个域环(rings)
- 如果 $j < i \Rightarrow D_i \subseteq D_j$
- 环0具有特权的全集



Protection

- **Multics的好处和限制**
  - 环/层次结构提供的不仅仅是基本的内核/用户或根/普通用户设计
  - 相当复杂 → 更多开销
  - 但不允许严格需要才知道
    - 对象在 D_j 中可访问，但在 D_i 中不可访问，那么必须是 $j < i$
    - 但是，每个在 D_i 中可以访问的段在 D_j 中也可以访问

Protection

- **访问矩阵**
  - 将保护视为矩阵（访问矩阵）
  - 行表示域
  - 列表示对象
  - **Access(i, j)** 是域 D_i 中执行的进程对对象 O_j 调用的操作集合

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	

j

i

访问矩阵的使用

- 如果域D_i中的进程试图对对象O_j执行“op”，则“op”必须在访问矩阵中
- 创建对象的用户可以为该对象定义访问列
- 可以扩展到动态保护
  - 添加、删除访问权限的操作
  - 特别访问权：
    - O_i的所有者
    - 将op从O_i复制到O_j（用“*”表示）
    - 控制 – D_i可以修改D_j访问权限
    - 迁移 – 从域D_i切换到D_j
  - 复制和所有者适用于一个对象（列）
  - 控制适用于域对象（把域当成对象）

访问矩阵的使用

- 访问矩阵设计将机制和策略分离
  - 机制
    - 操作系统提供访问矩阵+规则
    - 确保矩阵仅由授权代理操纵，并严格执行规则
  - 策略
    - 用户指定策略
    - 谁可以访问什么对象，以什么模式访问
- 但并不能解决一般的禁闭问题(confinement problem)
  - 保证对象最初持有的信息不能移到执行环境之外
  - 复制与所有者权限提供了一种机制，限制访问权限的传播。但没有提供适当工具，防止信息的传播（泄露）

Protection

- **以域为对象的访问矩阵**
  - switch ∈ access(i, j)：运行在 D_i 的进程允许切换到 D_j

domain \ object	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			

Protection

- **具有复制权限的访问矩阵**
  - 运行在 D_2 的进程可以把对 F_2 的 read 权限复制到 D_3, 这样 D_3 的进程也可对 F_2 进行 read 操作

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)

Protection

- **具有所有者权限的访问矩阵**
  - D_1 的进程可以在任意域增删 F_1 的任意权限；D_2 的进程可以在任意域为 F_2, F_3 增删任意权限

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

(a)

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

(b)

Protection

- **访问矩阵的控制权限**
  - 执行在 D_2 的进程，可以修改域 D_4 的访问权限

domain \ object	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			

访问矩阵的实现

- 通常，稀疏矩阵
- 方法1 – 全局表
  - 在表中存储有序的三元组 <域, 对象, 权限集>
  - 在域 D_i 中对对象 O_j 请求 M 操作 → 在表中查找 $<D_i, O_j, R_k>$
    - 使得 $M \in R_k$
  - 但表可能很大 → 无法放入主内存
  - 难以对对象进行分组（考虑一个所有域都可以读取的对象：所有域都要有一个read）

访问矩阵的实现

- 方法2 – 对象的访问列表
  - 每个列实现为一个对象的访问列表
  - 生成的每个对象列表由有序对<域, 权限集>组成，定义所有对该对象具有非空访问权限集的域
  - 易于扩展以包含默认集合 → 只要$M \in$ 默认集合，也允许访问
  - 每列 = 一个对象的访问控制列表
    - 定义哪些域可以执行什么操作

      域1 = 读、写
      域2 = 读
      域3 = 读

访问矩阵的实现

- 方法3 – 域的能力列表
  - 列表不是基于对象的，而是基于域的
  - 域的能力列表是对象及允许对其进行的操作的列表
  - 由其名称或地址表示的对象，称为能力(capability)
  - 对对象 O_j 执行操作 M，即进程请求操作并指定能力为参数
    - 拥有能力意味着允许访问
  - 能力列表与域关联，但不能被域中进程直接访问
    - 而是受保护对象，由操作系统维护并由用户间接访问
    - 能力作为一种“安全指针”，满足保护资源的需求
    - 此思想可扩展到应用程序级别的保护
  - 每行 = 能力列表（域作为关键字）
    - 对于每个域，允许对哪些对象执行哪些操作

      对象F1 – 读

      对象F4 – 读、写、执行

      对象F5 – 读、写、删除、复制

访问矩阵的实现

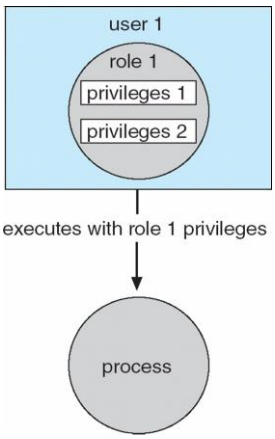
- 方法4 – 锁与钥匙
  - 访问列表和能力列表的折衷
  - 每个对象都有一个称为锁的唯一位模式的列表
  - 每个域都有一个称为钥匙的唯一位模式的列表
  - 域中的进程只有在域具有匹配对象中的一把锁的钥匙时，才能访问该对象

实现的比较

- 许多权衡考虑
  - 全局表很简单，但可能很大
  - 访问列表直接对应用户的需求
    - 确定非本地的域的访问权限集非常困难
    - 必须检查对对象的每次访问
      - 许多对象和访问权限 → 缓慢
  - 能力列表对定位给定进程的信息很有效
    - 但是撤回能力可能效率低下（需在系统中定位所有能力）
  - 锁与钥匙有效且灵活，钥匙可以自由地从一个域传递到另一个域，易于撤回
- 大多数系统使用访问列表和能力的组合
  - 首次访问对象 → 搜索访问列表
    - 如果允许，创建能力并附加到进程
      - 无需检查其他访问
    - 上次访问后，移除能力
    - 考虑文件系统中为每个文件设置ACL

访问控制

- 保护可应用于非文件资源
- Oracle Solaris 10提供了基于角色的访问控制(RBAC)以实现最低权限
  - 特权是执行系统调用或在系统调用中使用选项的权利
  - 可以分配给进程
  - 分配角色的用户将获授权访问特权和程序
    - 通过密码启用角色以获取特权
  - 类似于访问矩阵



Protection

- **撤回访问权限**
  - 删除域对对象的访问权限的各种选项
    - 立即vs.延迟
    - 可选vs.一般 （对于用户）
    - 部分vs.全部
    - 临时vs.永久
  - 访问列表 – 从访问列表中删除访问权限
    - 简单 – 搜索访问列表并删除条目
    - 立即，一般或可选，全部或部分，永久或临时

Protection

- **撤回访问权限**
  - 能力列表 – 被撤回前，需在系统中定位能力
    - 重新获得 – 定期删除。如果撤回，则进程重新获取时将被拒绝
    - 反向指针 – 从每个对象到该对象所有能力的指针集(MULTICS)
    - 间接 – 能力指向全局表条目，条目指向对象。撤回时从全局表中删除条目，非可选撤回
    - 钥匙 – 与能力关联的唯一位模式，在创建能力时生成
      - 主钥与对象关联，钥匙与主钥匹配时可访问
      - 撤回 – 创建新的主钥
      - 策略决定谁可以创建和修改钥匙
        - 对象所有者还是其他人？

Protection

- **基于能力的系统**
  - 例子：Hydra系统
    - 系统已知和解释一套固定的可能访问权限集
      - 例如，读、写或执行每个内存段
      - 用户可以声明其他辅助权限，并向保护系统注册
      - 访问进程必须具备能力并知道操作名称
      - 特定类型的可信程序允许权限扩充（超过调用程序持有的权限）
    - 用户定义权限的解释仅由用户程序进行；但系统为使用这些权限提供访问保护
    - 对对象的操作由过程定义 – 过程是由能力间接访问对象
    - 解决双向怀疑子系统问题（用户调用一个服务程序）
    - 包括预先编写的安全例程库

Protection

- **基于能力的系统**
  - 剑桥CAP系统
    - 简单但强大
    - 数据能力 – 提供与对象相关的单个存储段的标准读、写、执行操作 – 在微码中解释
    - 软件能力 – 由子系统的保护（即特权）程序解释
      - 只能访问自己的子系统
      - （没有提供库）程序员必须学习保护原理和技术

Protection

- **基于语言的保护**
  - 编程语言中的保护规范允许对资源分配和使用的策略进行高级描述
  - 当硬件支持的自动检查不可用时，语言实现可提供软件来实施保护
  - 对硬件和操作系统提供的任何保护系统，通过解释保护规范来生成相应的调用

Protection

- **Java 2中的保护**
  - 保护由Java虚拟机(JVM)处理
  - 当一个类class被JVM加载时，它将被分配一个保护域
  - 保护域指示类可以（和不能）执行的操作
  - 如果调用了一个执行特权操作的库方法，则会检查堆栈以确保库可以执行该操作
  - 通常，Java的加载时和运行时检查会强制要求类型安全
  - 类有效地封装并保护数据和方法不受其他类的影响

Protection

- **堆栈检测**
  - 每个JVM线程都有一个关联堆栈，包含正在进行的方法调用
  - doPrivileged代码块：标记拥有权限，不被信任的调用者可直接或间接访问保护资源
  - checkPermissions()：执行检查堆栈，是否有doPrivileged标记
    - gui的open没有doPrivileged，不被允许！

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	get(url);
	open(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission (a, connect);
			connect (a);
			...