[操作系统安全机制 1](#_Toc105411445)

[review 1](#_Toc105411446)

[基于硬件的进程内隔离：Enclave 2](#_Toc105411447)

[EPC(Enclave Page Cache) 3](#_Toc105411448)

[资源隔离 3](#_Toc105411449)

[小结：隔离的不同方式 8](#_Toc105411450)

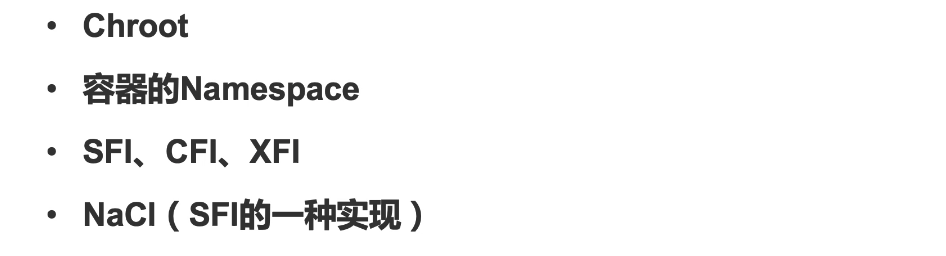
[安全 8](#_Toc105411451)

[安全的三个概念 10](#_Toc105411452)

# 操作系统安全机制

## review

上节课我们讲了轻量级隔离，这已经是OS常用的一种安全方式了。



要做深层次隔离，一定要namespace，要改内核，比如ip，userid，pid都要隔离开。你的pid100和我的pid100就是两个pid。

SFI，CFI，XFI不是由内核直接提供，SFI是由编译器提供的，每个load store之前加个check。这个有点像java，但是粒度更粗。只有SFI不够，因为依赖顺序执行，如果跳过检查直接到load store，所以要CFI防止程序跳来跳去，结合在一起形成XFI。如果把XFI用在驱动隔离上面的话，这就能大大增加kernel的安全性。如果能够把driver很好地隔离起来就能很好的提升kernel的健壮性。

NaCL是SFI的一种实现，在浏览器中用的一种方式。包括现在web assembly能用一个虚拟机的方式去运行。直接在语言级做隔离。

事物发展的规律，尤其是一开始有一些功能是在内核里做的，后来慢慢发展想是不是能放到用户态。放到用户态轻量级了，跟底下的kernel的版本没关系了。所以如果能把一些机制性的东西变到用户态，之后会变成一个lib。而且自己如果要升级不用到kernel里面说：帮我升级一下吧，帮我整合到主线吧。

serverless经历这样一个典型的过程：一开始用虚拟化，然后用docker，现在在考虑用web assembly。

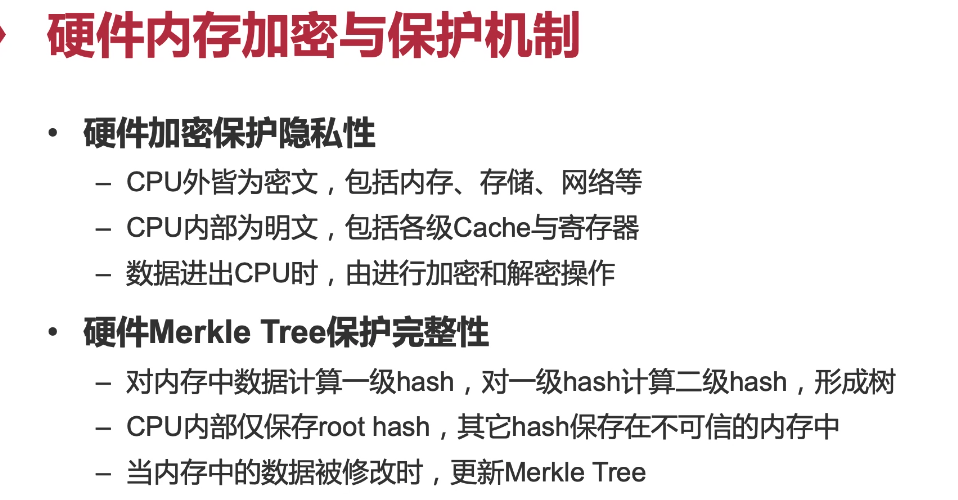
这是用软件的方法做隔离的思路。还有一种就是hardware。

## 基于硬件的进程内隔离：Enclave

我们前面提到的一些关于OS的隔离，硬件能不能直接提供小隔间。kernel一向的思路：把自己的功能压缩压缩，把功能往应用态放。



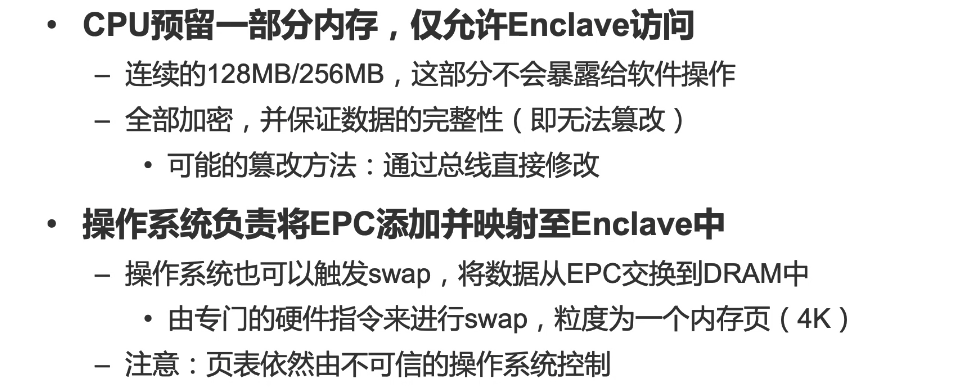
15年的时候引入。关键的技术是hardware提供了一个很小的运行环境，与外部完全是隔离的。只保护内部的东西，外部的所有其他代码在他看来都是不可信的。



Merkle Tree是hash的方式，整个内存算一个hash。可以把一部分的hash直接保存在CPU里面，当我去读某一块数据的时候，把周围的hash都算一遍。

用硬件的方式保护内存（软件是做不到的，因为可以通过直接拔走冷冻之类的方式）

### EPC(Enclave Page Cache)



CPU内部是要知道enclave这个概念的。enclave创建之后，CPU里面要记录一些数据结构表示这个enclave。里面要记录：这个enclave拿了几块EPC（128M中的几块EPC），运行着发现内存不够了可以swap。



4k，4k的加，metadata和coda，data一起加到enclave里面。

enclave里面的calling convention和外面都可能有差别。

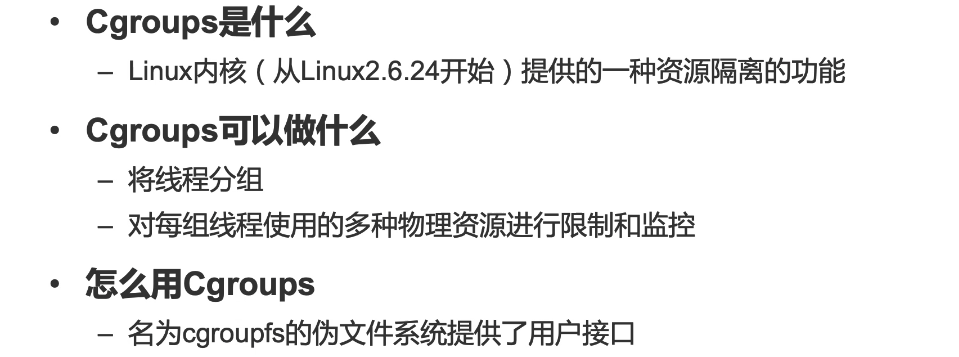
256M太少了，后来intel推出了支持64G及以上的空间。

虚拟化为隔离，容器隔离（容器里可能有很多进程），进程隔离，进程内部隔离（软件两种：CFI+SFI，硬件方法：Enclave）

前面说的都是安全性隔离。把你放在一个sandbox里面出不来。

## 资源隔离

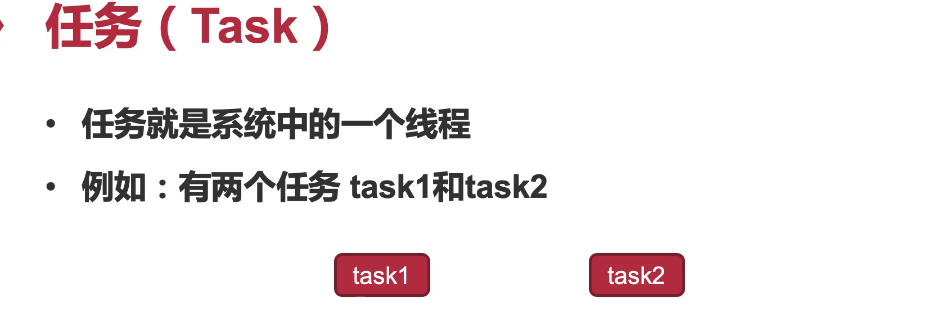
资源竞争问题。Cgroups做的就是这样一个东西。



一直到现在发展了好几代了还是不够成熟。

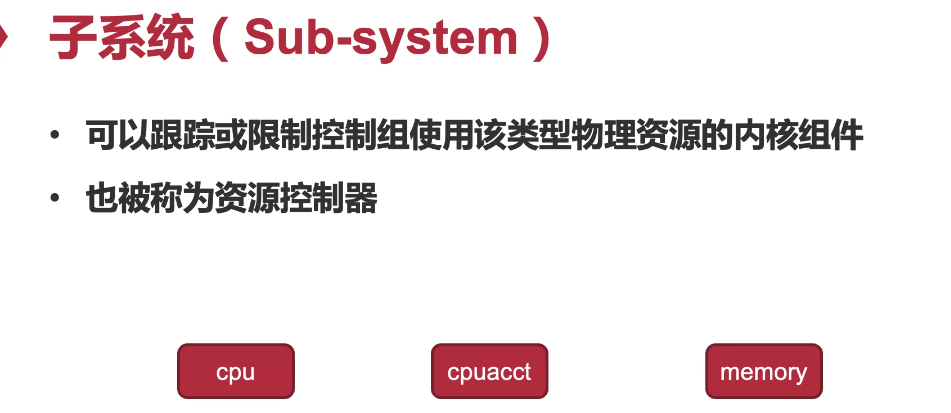
1. cgroups对线程分组。

2. 对每组线程使用的物理资源进程限制和监控。

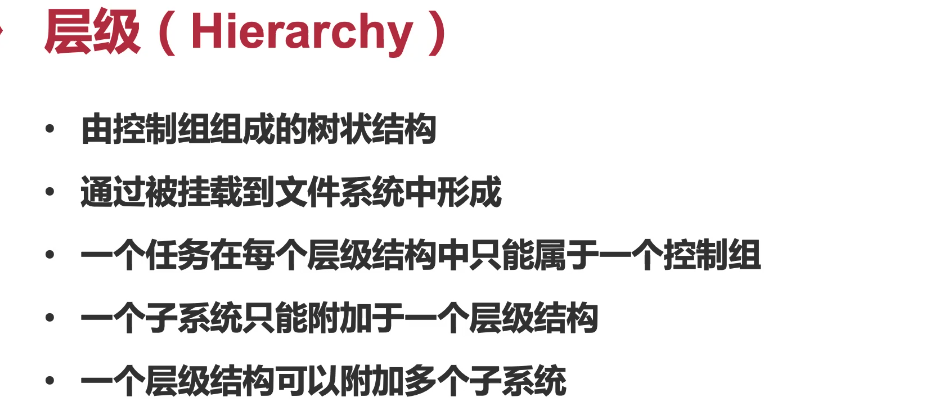


线程是执行的基本单位，虽然资源分配是以进程为粒度的。如果最小粒度是线程，进程肯定也能处理，但是最小粒度是进程时，线程就没有办法。



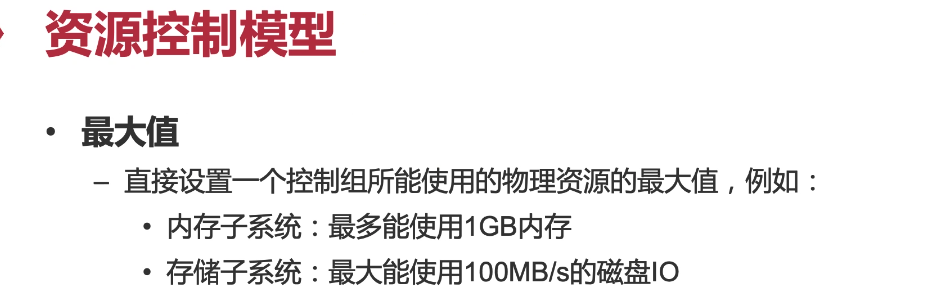


控制组组成了树状结构，挂载到文件系统里。



一些tricky：内存和CPU完全有关系肯定不对，但是完全没关系吧，也不对。从实际角度来看是case by case。解耦不一定是好事：汽车有四个方向盘。

amazon之类的在提供serverless的时候，提供的资源组合就是内存，一块内存对应多少CPU，你可以通过内存的量去决定，多少内存绑定一个CPU call，不用考虑网络带宽，CPU之类的该怎么配。



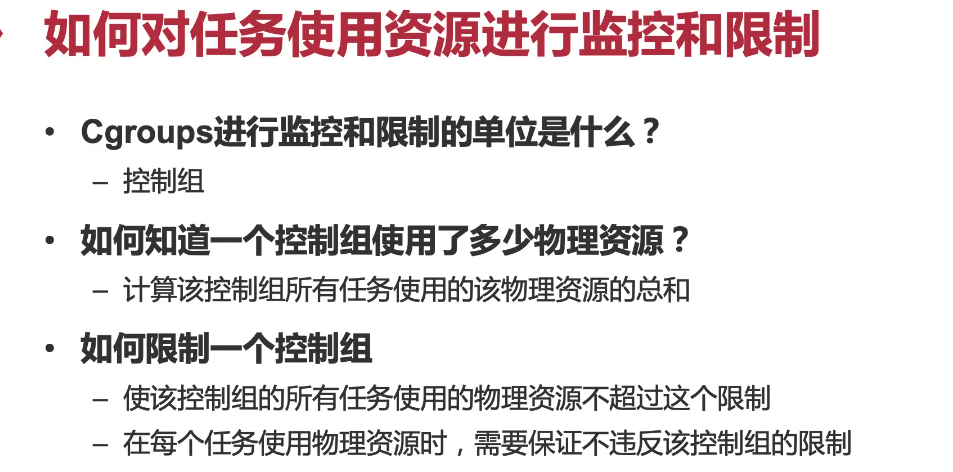
对于每一个资源控制模型，比如最大值，某个cgroup只能访问1G的内存，这个比较容易做，因为1G内存比较容易测量。这里说的是物理内存。但物理内存不是要就给的。每次当一个应用用物理内存时，os要调alloc\_page，通过buddy system。在用buddy\_system的时候os就可以在这里做监控（加个计数器），超过1G就不能调alloc page了，只能swap。而且这个LRU算法只能在属于自己的内存里去做。对于IO来说，不那么简单了，内存只要在一个地方卡住就可以了。因为你要监控动态速度，向硬件发出实际请求的时候。谁发了多大的请求，时间频率是多少。比如第一秒发了200，那么第二秒就不能发，平均下来就是100了。但这样并没有卡在100之内，时间间隔可以更小一点。如果说你是以1s去做，你的粒度就比100s要粗一点。但是如果以1ms为粒度，你发的就会慢一点。

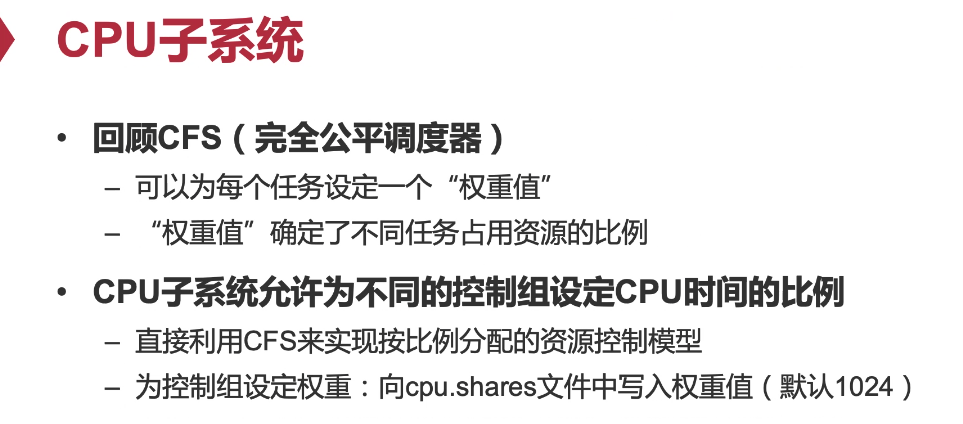
假设要发100M数据，以1s作为粒度，发了100ms就发了100M了，那么后面900ms就不发了，但是只看前100ms你的速度是1G。

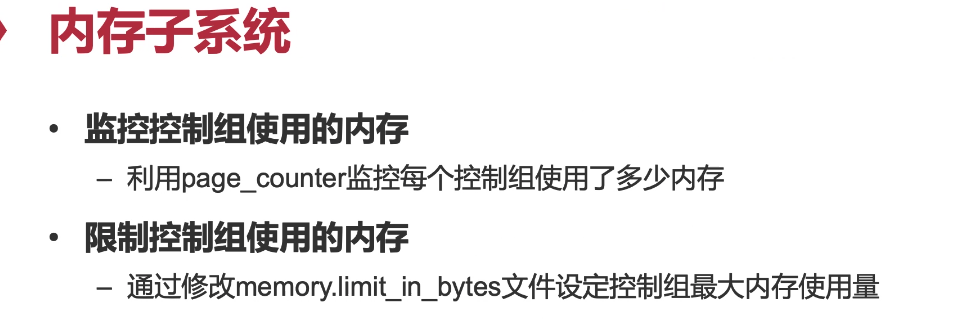


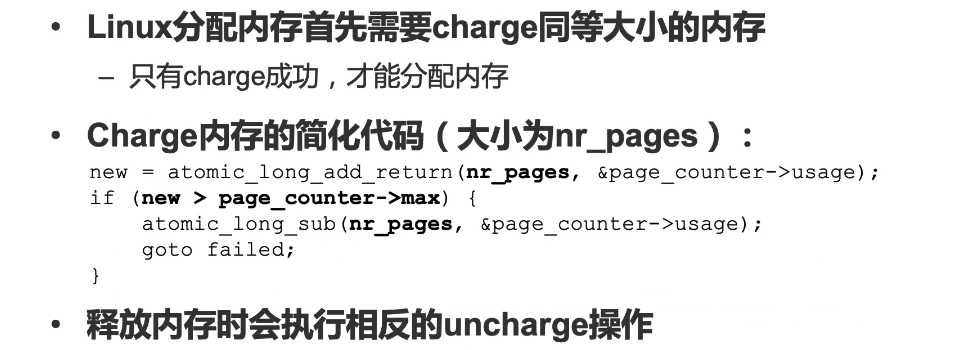
如果卡死用最大值的话，一个人就会有意见，我一个人用为什么不让我用满。

所以两个人就是250：250，一个人就是500。



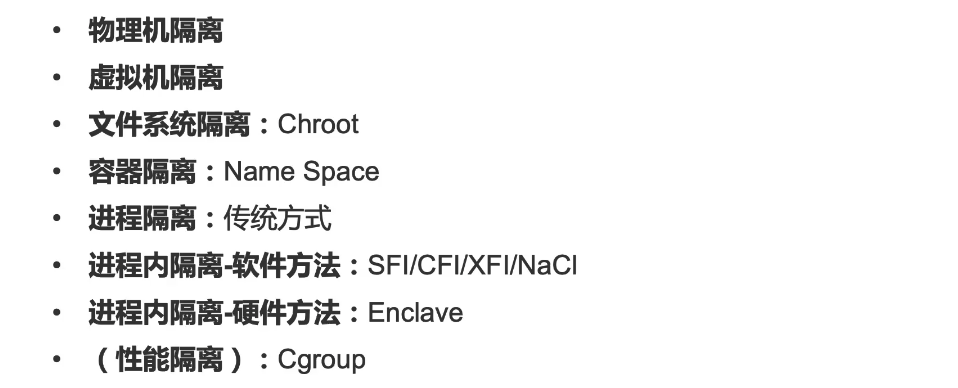






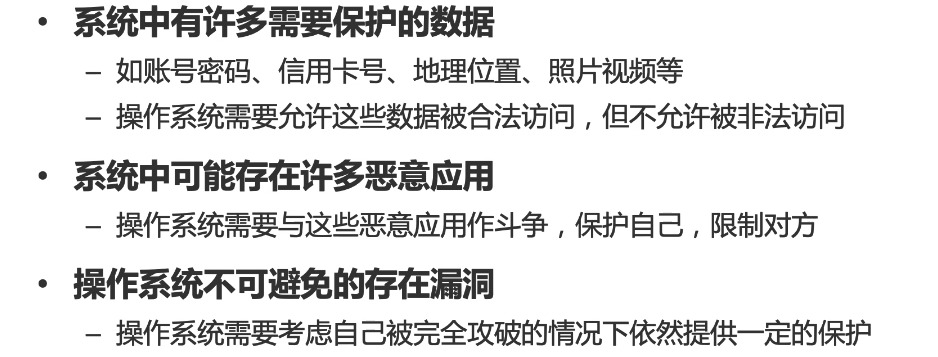
charge的时候会更新这个counter。

### 小结：隔离的不同方式



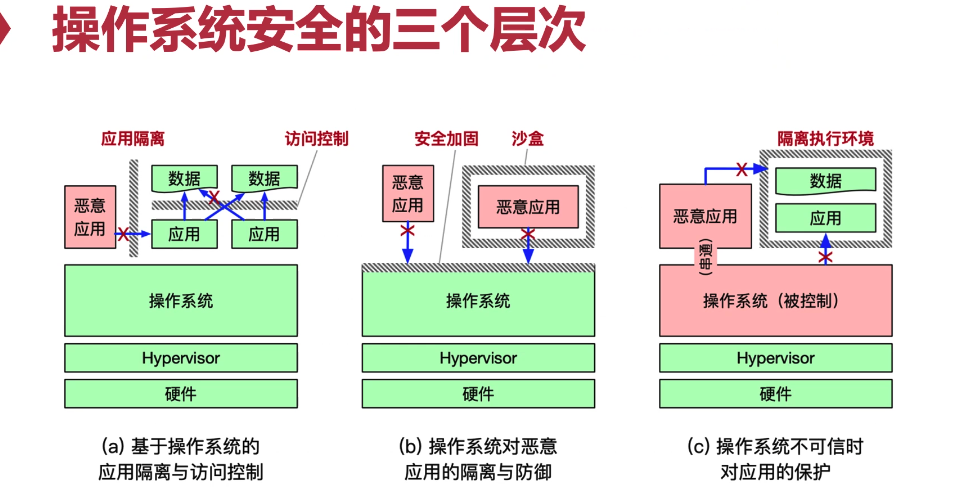
# 安全

安全是操作系统的重要功能和服务



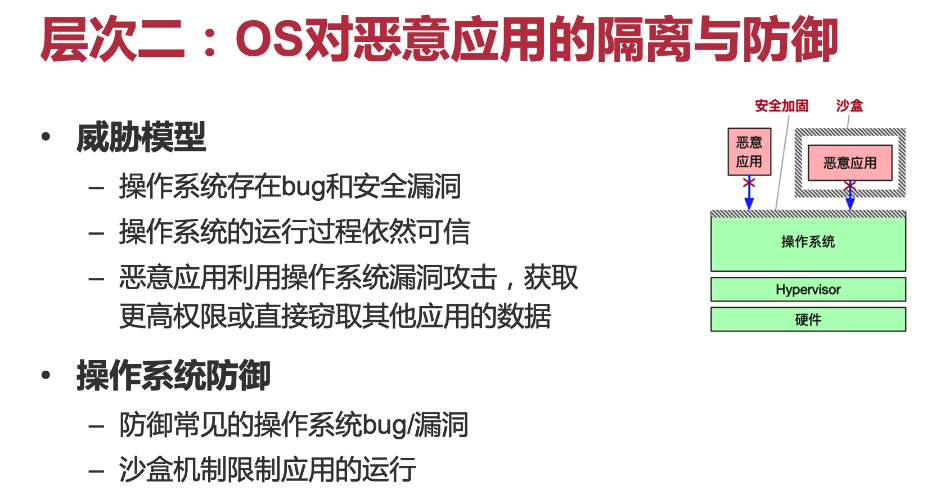
被合法访问和不允许非法访问不是冗余的。后者是很难去证明100%成立的。

OS自己要去抵御恶意应用程序。OS漏洞不可避免，如果保证自己被完全攻破了依然提供保护。

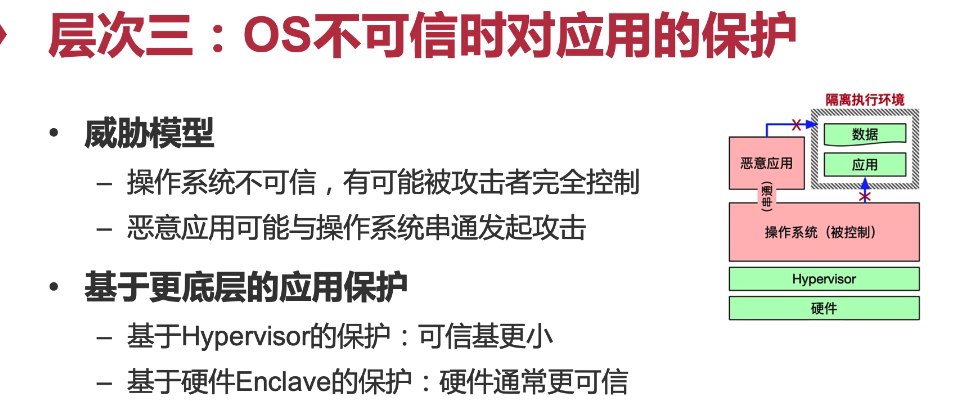


OS在启动的时候还好没啥问题。做好之后后面运行的时候出问题了可以靠硬件。



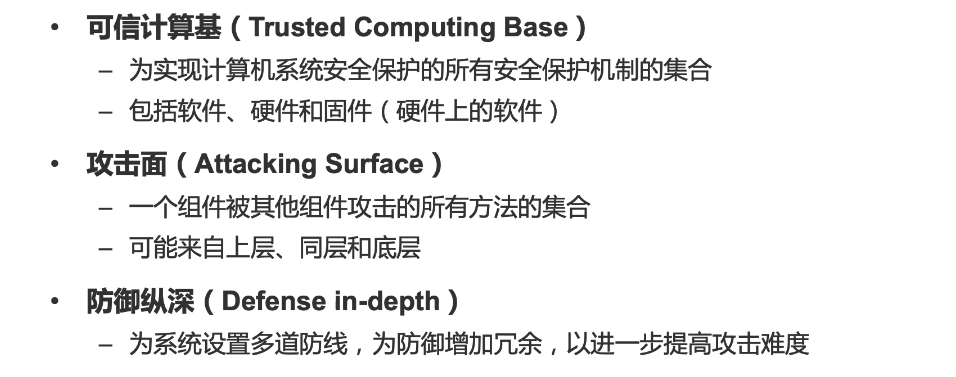


较少使用的syscall可能有bug，给应用设filter，只能用有限的系统调用——减小攻击面。



硬件提供的新抽象。

### 安全的三个概念



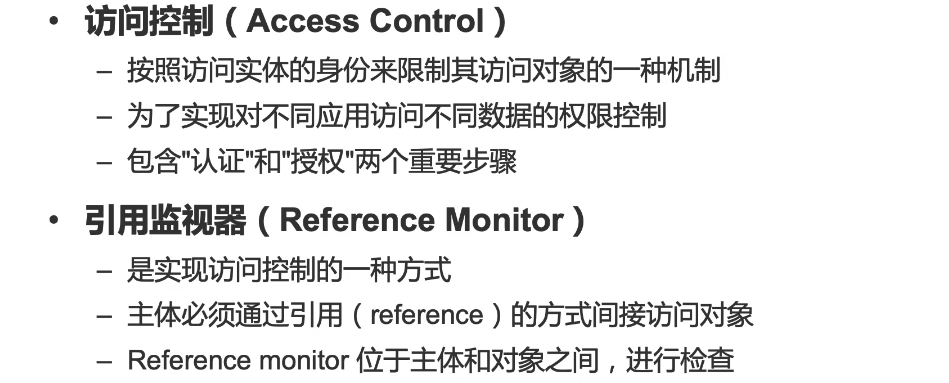
攻击面如果是300个syscall和30个syscall 3个syscall肯定是不一样的。还有可能来自于同层。底层：有人直接把你内存拔了。hypervisor有bug怎么办？

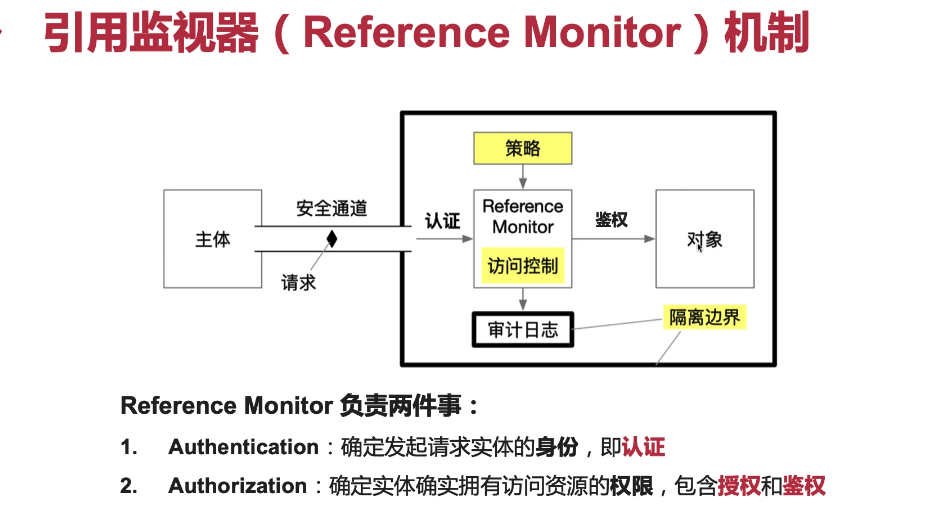
可信计算基越小越好，攻击面越少越好，防御纵深越深越好。

还有trade off，要考虑可用性，性能等等。

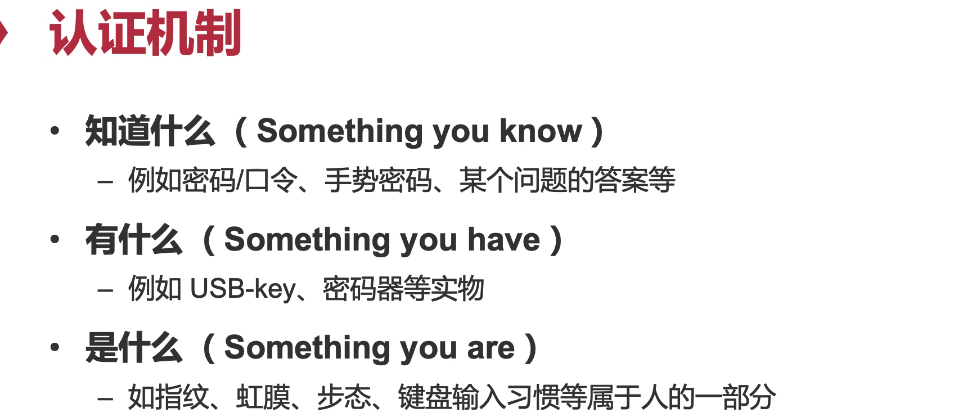
今天讨论第一个机制：OS可信

访问控制与引用监视器



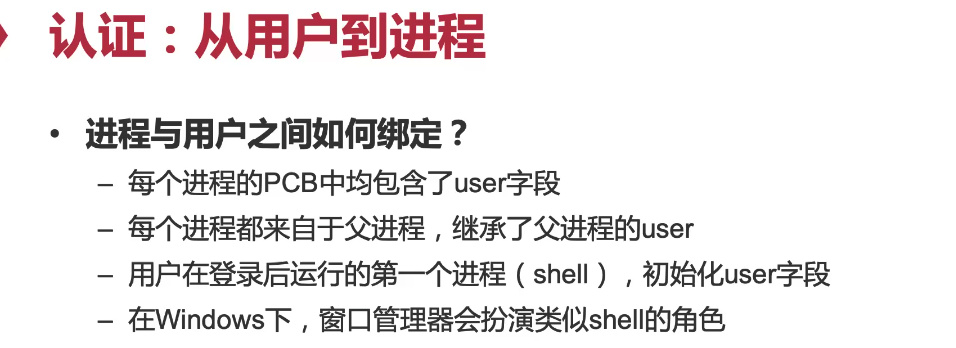


一个进程想要访问一个文件，不能直接到磁盘，要经过os的reference monitor。



无密码登录，有密码是最简单的，因为你知道是一个很tricky的东西。（你知道别人也可以知道，而且你不知道别人知不知道）

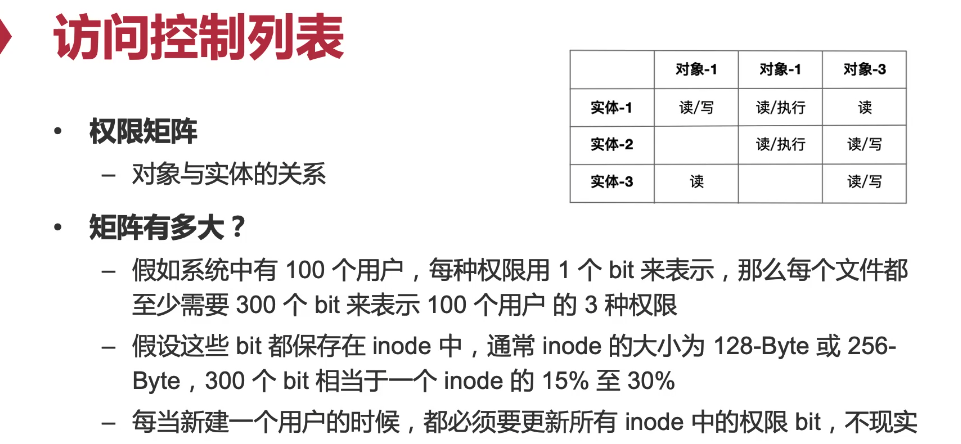
apple，google，facebook在解决无密码登录这个问题。



这个userid如何初始化：继承自父进程的id。

线程hierarchy的原因：zombie，继承一些信息。

windows：窗口管理器

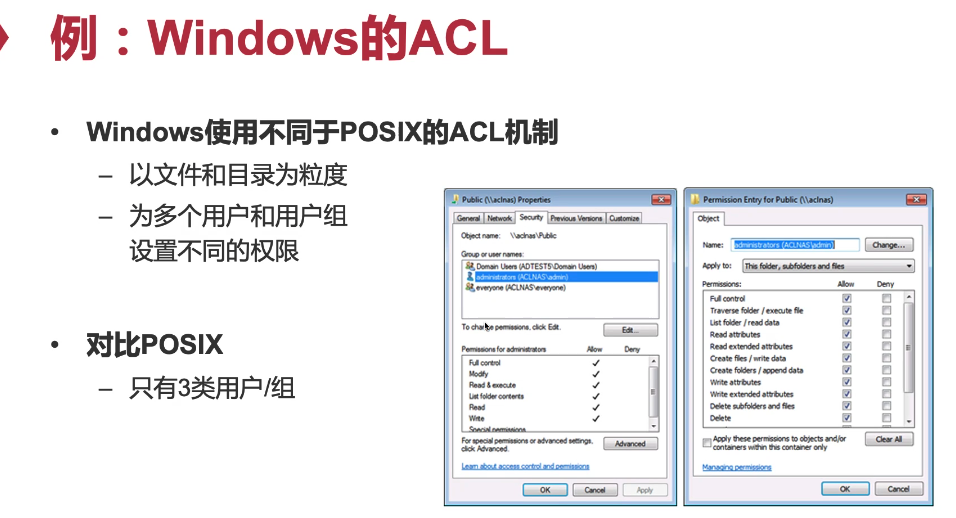


实体：进程1，进程2，进程3

每种权限有一个bit表示yes or no。

因为2，3 ——> 不现实

POXIS文件权限机制，9个bit：可读可写可执行 ✖️ 三种用户（owner，owner group，others）

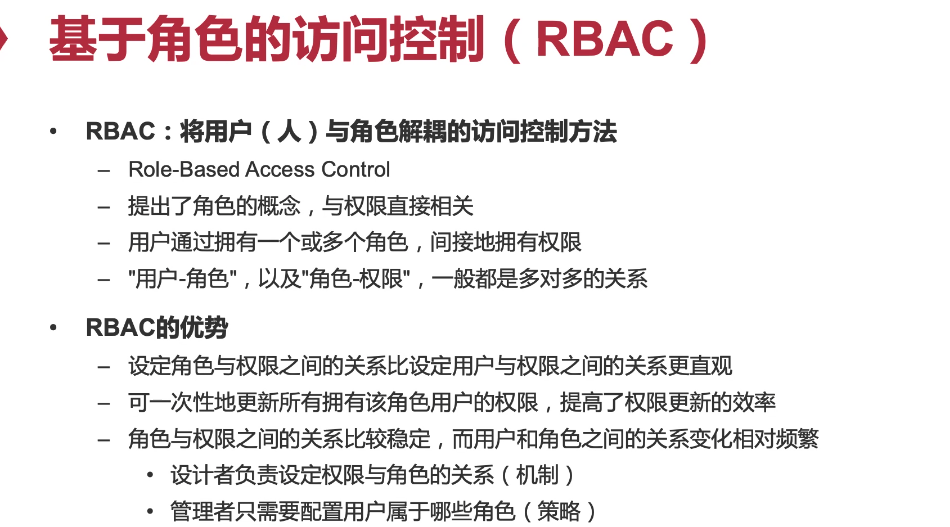


默认属于owner，他可以通过edit加入更多的，可以给不同的group设置不同的权限。

权限比3个要多得多。所以9个bit不够用。肯定更多：权限种类 ✖️ 用户数

记在ntfs的attribute里面。

windows不支持POSIX的ACL这是一个很重要的不兼容。

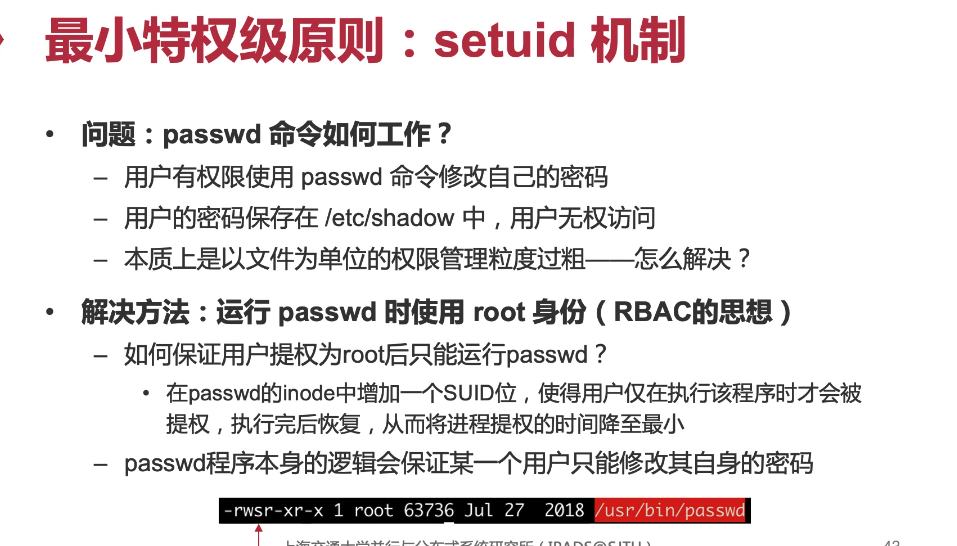


OS用的并不是RBAC而是ACL，但是OS也可以通过设置group来提供这样的机制。

比如linux有admin。

RBAC不能同时是什么，每时每刻只有一个角色。这样就不能用一个bash打包三个角色才能访问到的文件。简化版的RBAC就是把角色用group来实现（比如linux），但不是真正的RBAC。

setuid

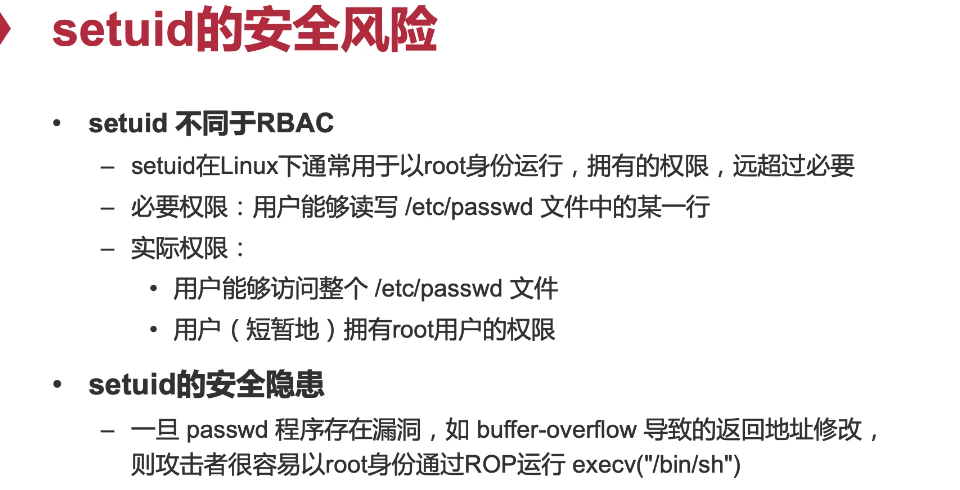


windows是每个人密码放不同的文件里。

因为是一整个文件，所以做不到只改自己的那一行。

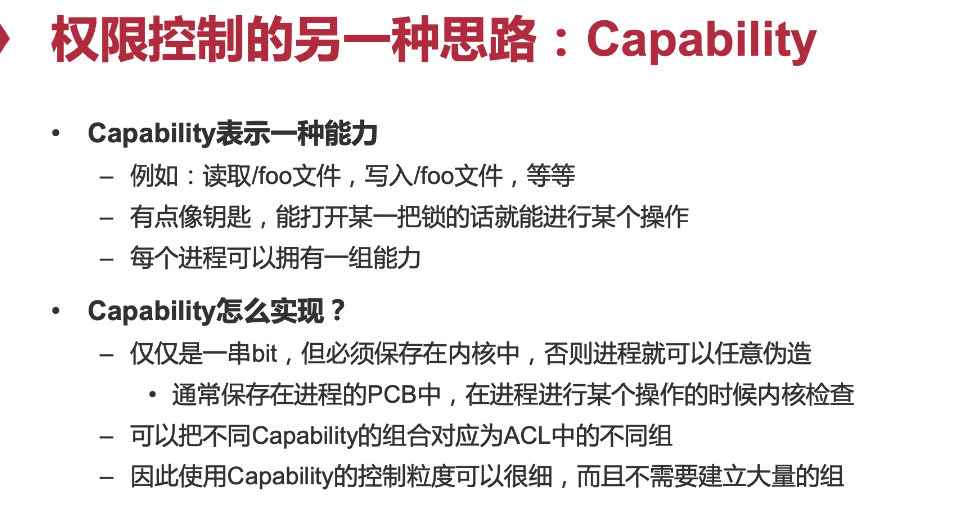
password是二进制，它的owner是root，X位变成S，如何保证这段代码只改passwd文件中的某一行呢？这就由passwd自身的代码逻辑来保证。

一般setuid是以root运行，所拥有的权限远远超过必要。



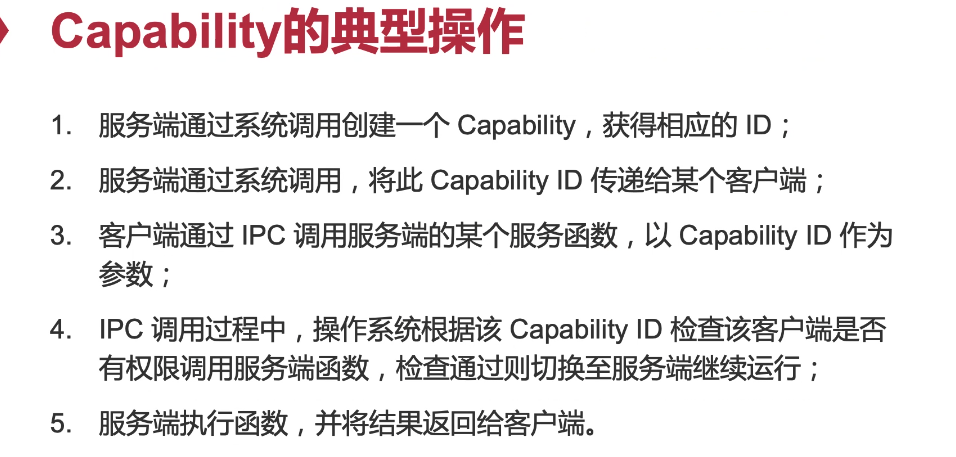
拿到了具有root权限的shell。

整个过程都在赌passwd没有漏洞。passwd是我们的TCB（trust computer base）

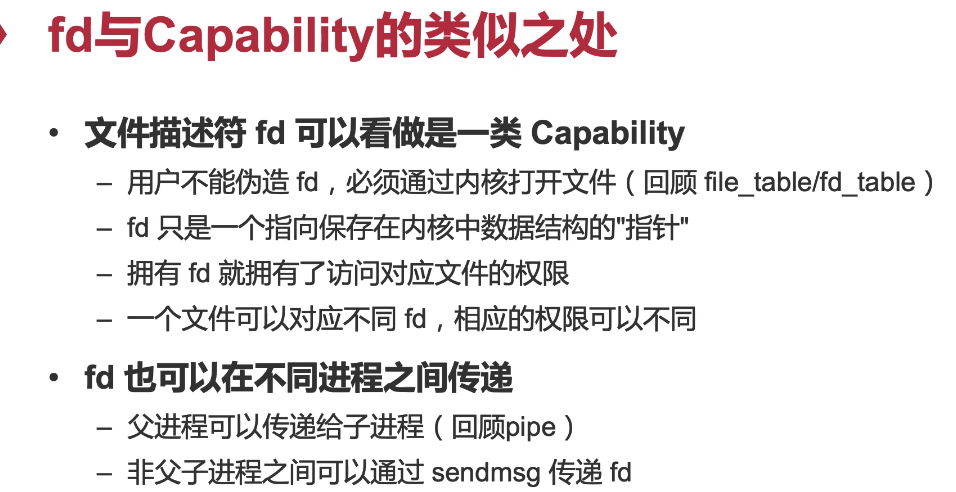


与ACL类似。ACL类似一个房间门口有一个大汉手里有一张list。

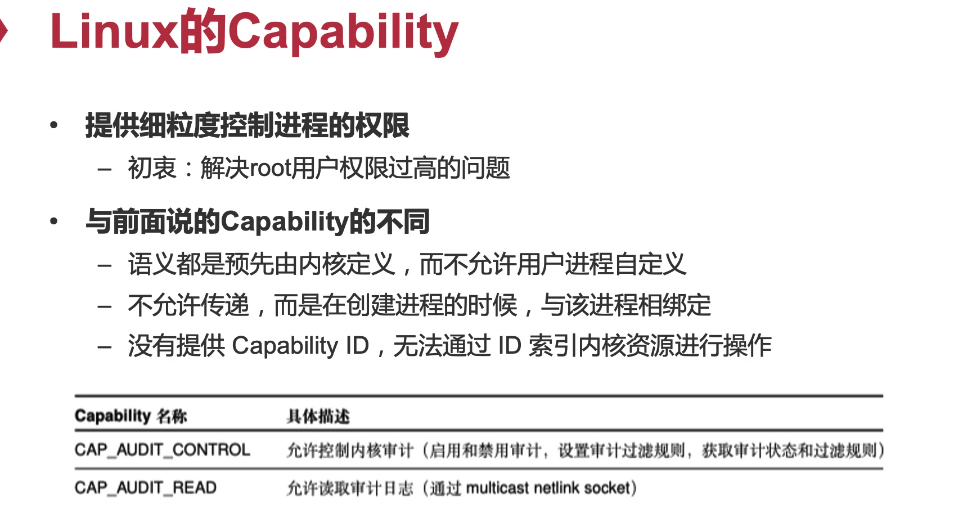
Capability类似钥匙，但是不能被伪造——放在内核里（不然用户复制一下就可以了）有了capability之后粒度就可以更细。



2钥匙传递这就和ACL不一样了，ACL身份不能传递。



sendmsg拿到fd之后也可以访问



不能自己创建。linux的capability和我们前面说的capability是两个事情。chcore中使用的前面所说的capability，所有的资源都抽象成capability。