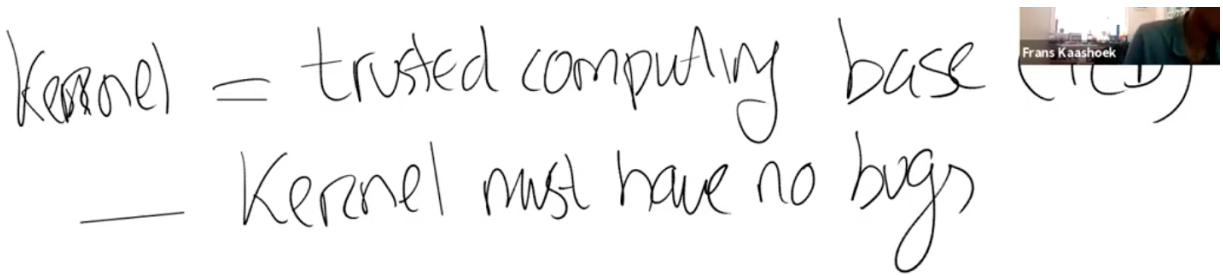
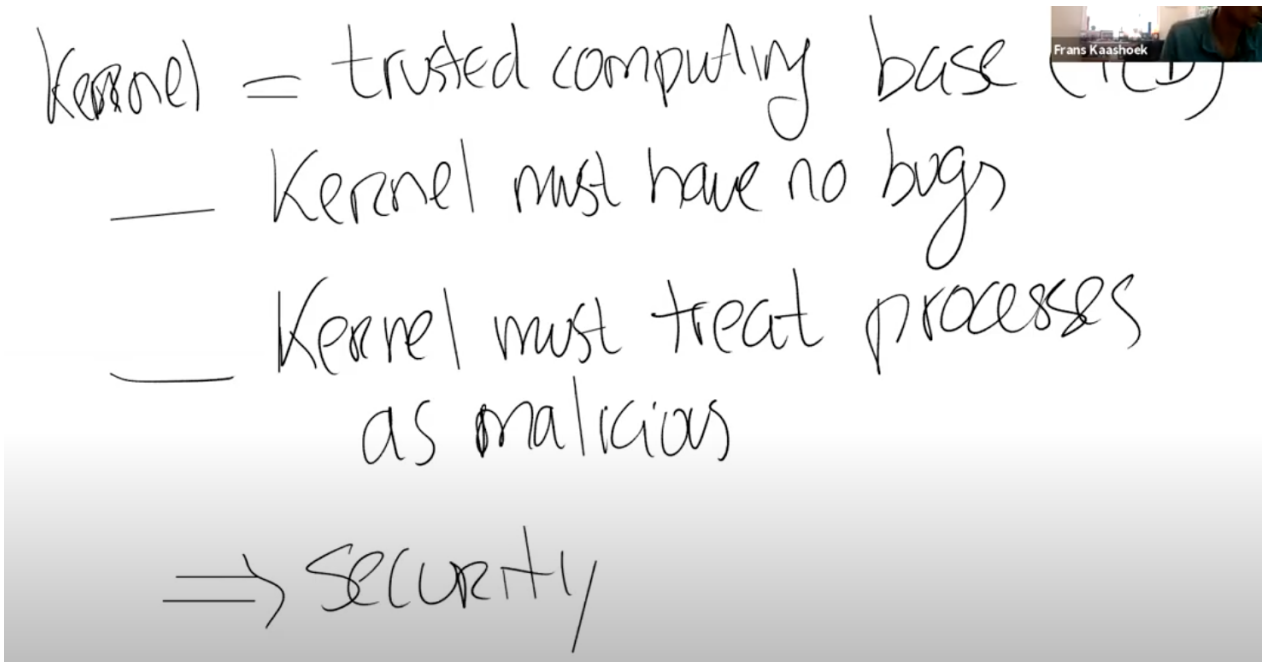
3.6 宏内核 vs 微内核 （Monolithic Kernel vs Micro Kernel）

现在，我们有了一种方法，可以通过系统调用或者说ECALL指令，将控制权从应用程序转到操作系统中。之后内核负责实现具体的功能并检查参数以确保不会被一些坏的参数所欺骗。所以内核有时候也被称为可被信任的计算空间（Trusted Computing Base），在一些安全的术语中也被称为TCB。

基本上来说，要被称为TCB，内核首先要是正确且没有Bug的**。**假设内核中有Bug，攻击者可能会利用那个Bug，并将这个Bug转变成漏洞，这个漏洞使得攻击者可以打破操作系统的隔离性并接管内核。所以内核真的是需要越少的Bug越好（但是谁不是呢）。

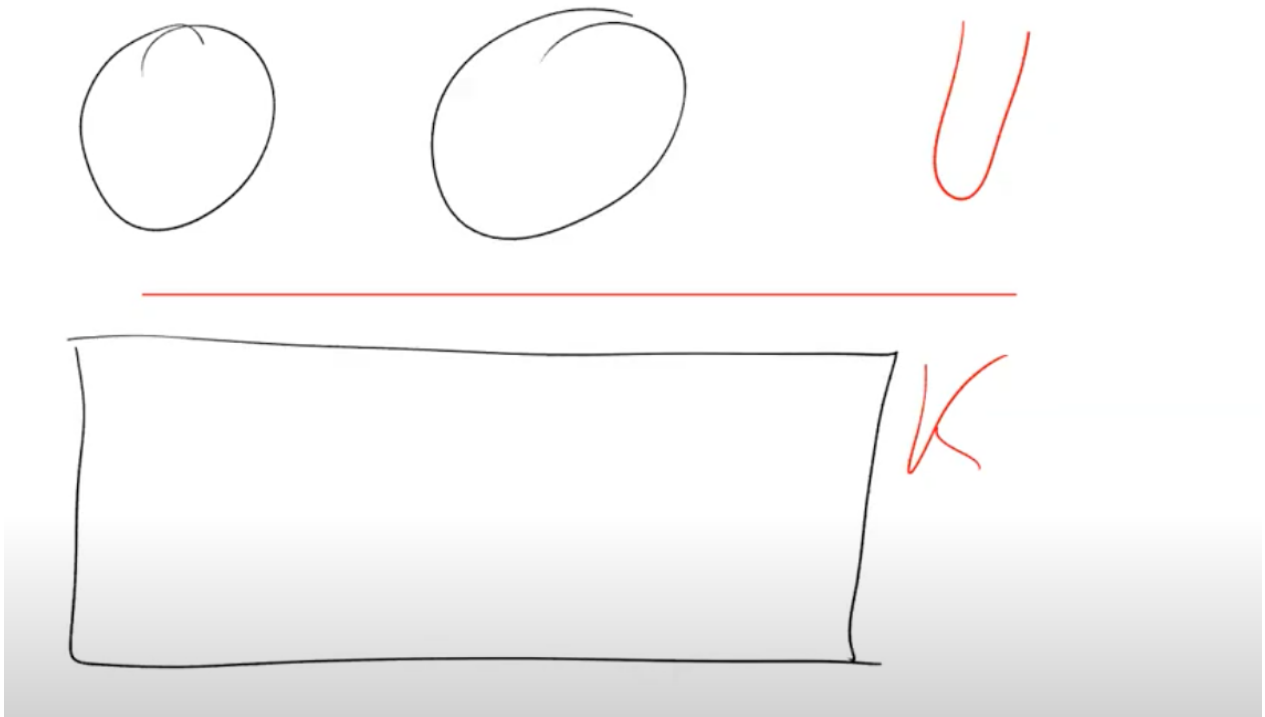


另一方面，内核必须要将用户应用程序或者进程当做是恶意的。如我之前所说的，内核的设计人员在编写和实现内核代码时，必须要有安全的思想。这个目标很难实现，因为当你的操作系统变得足够大的时候，很多事情就不是那么直观了。你知道的，几乎每一个你用过的或者被广泛使用的操作系统，时不时的都有一个安全漏洞。就算被修复了，但是过了一段时间，又会出现一个新的漏洞。我们之后会介绍为什么很难让所有部分都正确工作，但是你要知道是内核需要做一些tricky的工作，需要操纵硬件，需要非常小心做检查，所以很容易就出现一些小的疏漏，进而触发一个Bug。这也是可以理解的。

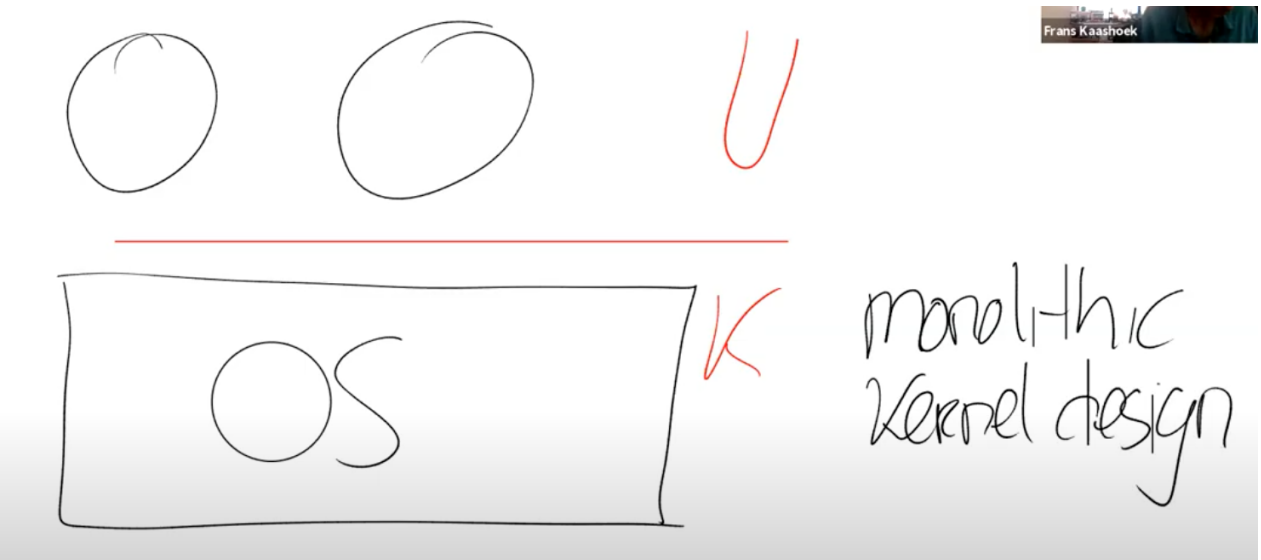


一个有趣的问题是，什么程序应该运行在kernel mode？敏感的代码肯定是运行在kernel mode，因为这是Trusted Computing Base。

对于这个问题的一个答案是，首先我们会有user/kernel边界，在上面是应用程序，在下面是运行在kernel mode的程序。



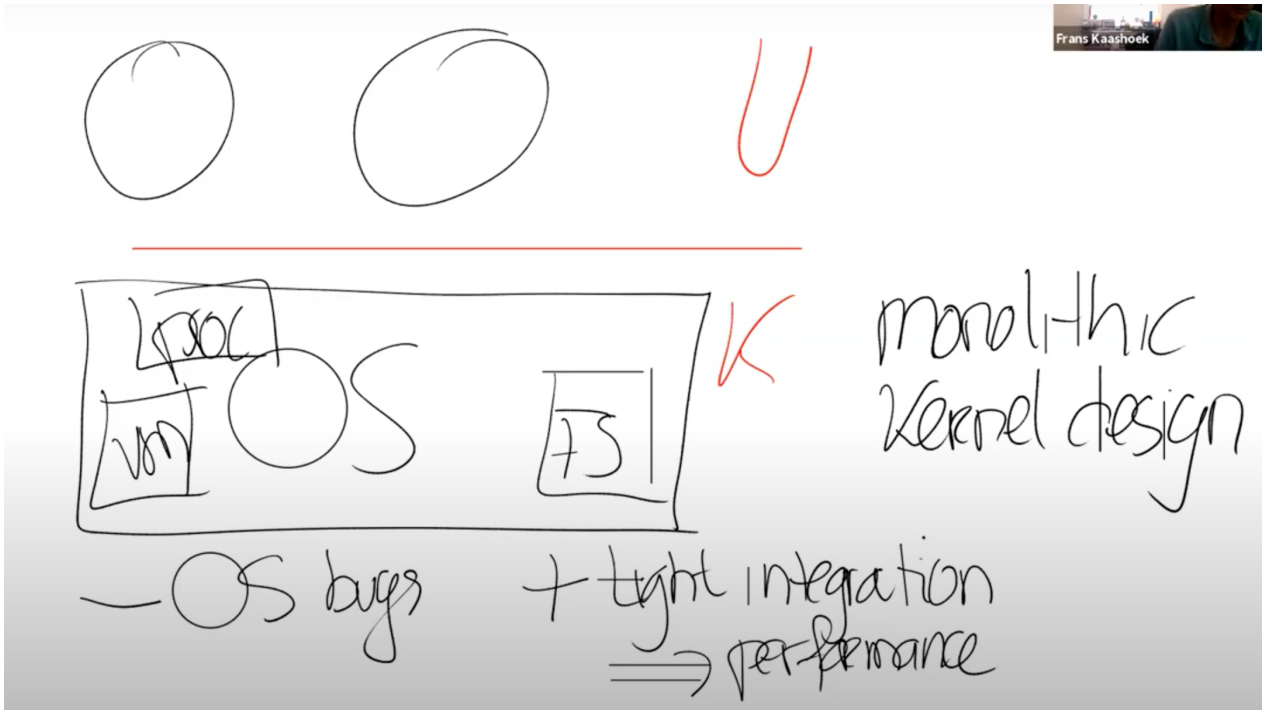
其中一个选项是让整个操作系统代码都运行在kernel mode。大多数的Unix操作系统实现都运行在kernel mode。比如，XV6中，所有的操作系统服务都在kernel mode中，这种形式被称为Monolithic Kernel Design（宏内核）。



这里有几件事情需要注意：

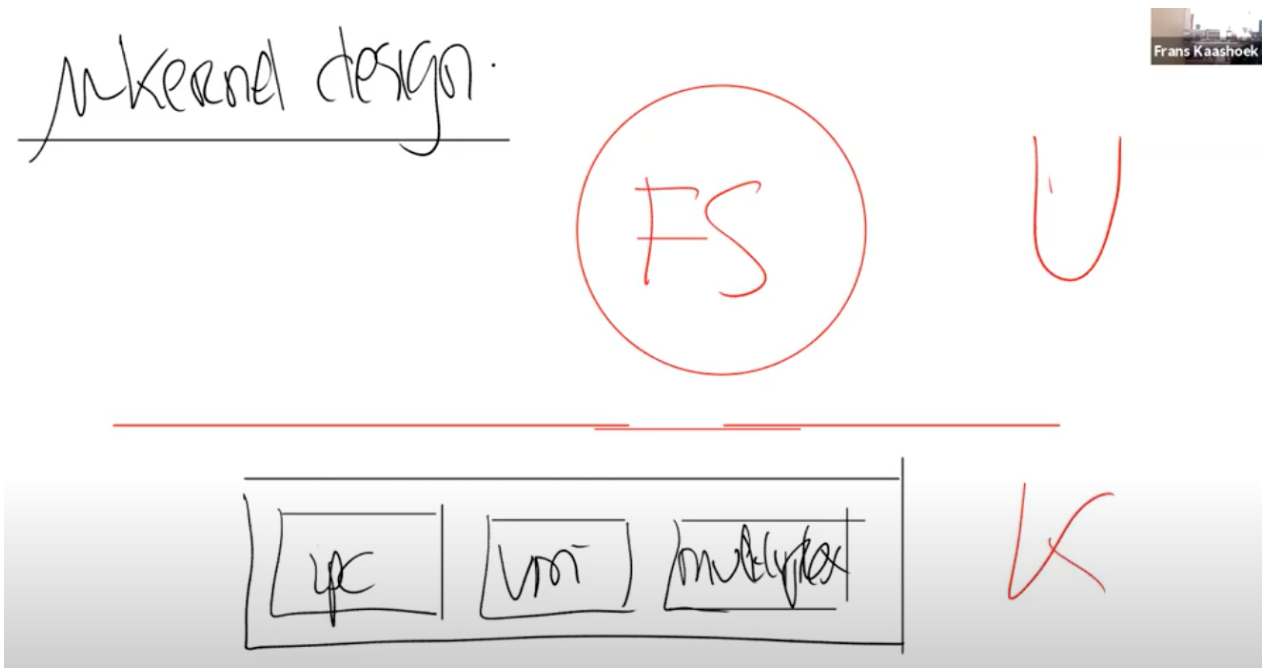
- 首先，如果考虑Bug的话，这种方式不太好。在一个宏内核中，任何一个操作系统的Bug都有可能成为漏洞。因为我们现在在内核中运行了一个巨大的操作系统，出现Bug的可能性更大了。你们可以去查一些统计信息，平均每3000行代码都会有几个Bug，所以如果有许多行代码运行在内核中，那么出现严重Bug的可能性也变得更大。所以从安全的角度来说，在内核中有大量的代码是宏内核的缺点。
- 另一方面，如果你去看一个操作系统，它包含了各种各样的组成部分，比如说文件系统，虚拟内存，进程管理，这些都是操作系统内实现了特定功能的子模块。宏内核的优势在于，因为这些子模块现在都位于同一个程序中，它们可以紧密的集成在一起，这样的集成提供很好的性能。例如Linux，它就有很不错的性能。

上面是对于内核的一种设计方式。

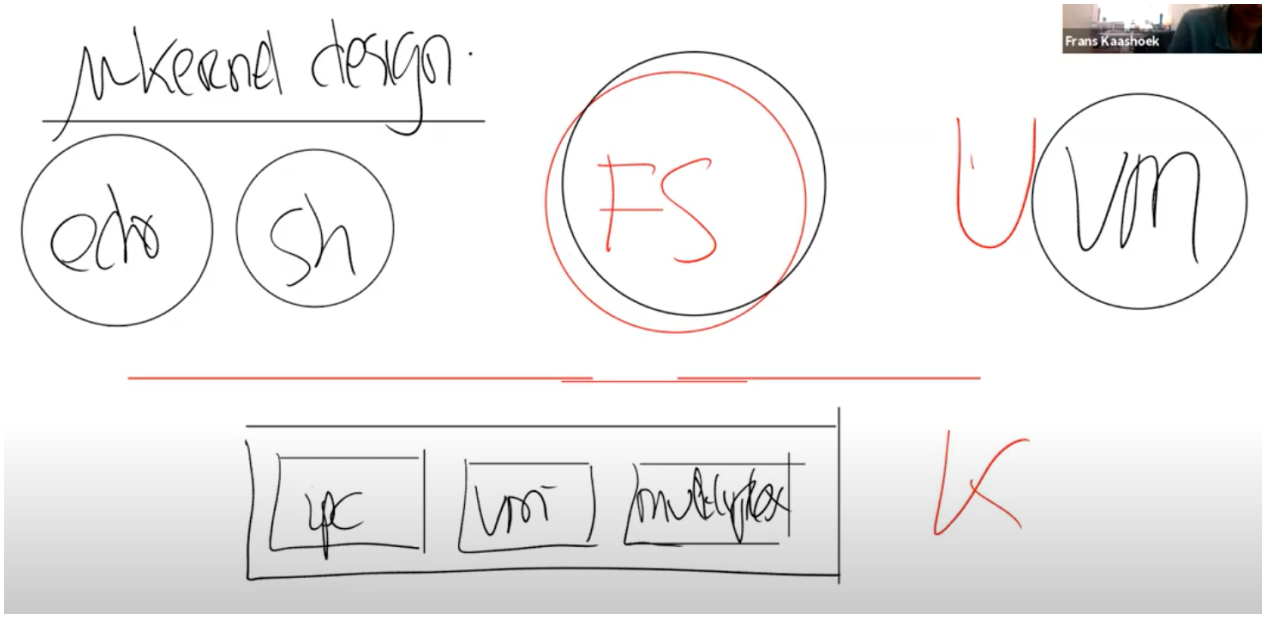


另一种设计主要关注点是减少内核中的代码，它被称为Micro Kernel Design（微内核）。在这种模式下，希望在kernel mode中运行尽可能少的代码。所以这种设计下还是有内核，但是内核只有非常少的几个模块，例如，内核通常会有一些IPC的实现或者是Message passing；非常少的虚拟内存的支持，可能只支持了page table；以及分时复用CPU的一些支持。

微内核的目的在于将大部分的操作系统运行在内核之外。所以，我们还是会有user mode以及user/kernel mode的边界。但是我们现在会将原来在内核中的其他部分，作为普通的用户程序来运行。比如文件系统可能就是个常规的用户空间程序。这个文件系统我不小心画成了红色，其实我想画成黑色的。

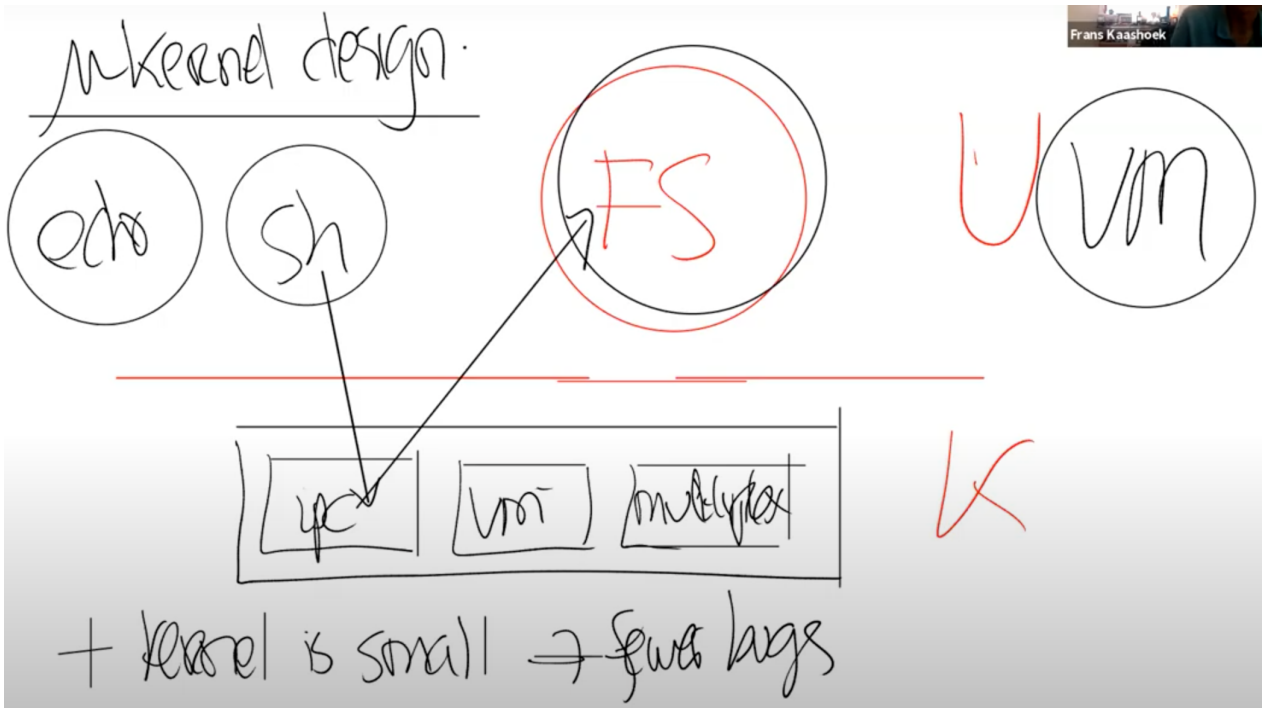


现在，文件系统运行的就像一个普通的用户程序，就像echo，Shell一样，这些程序都运行在用户空间。可能还会有一些其他的用户应用程序，例如虚拟内存系统的一部分也会以一个普通的应用程序的形式运行在user mode。

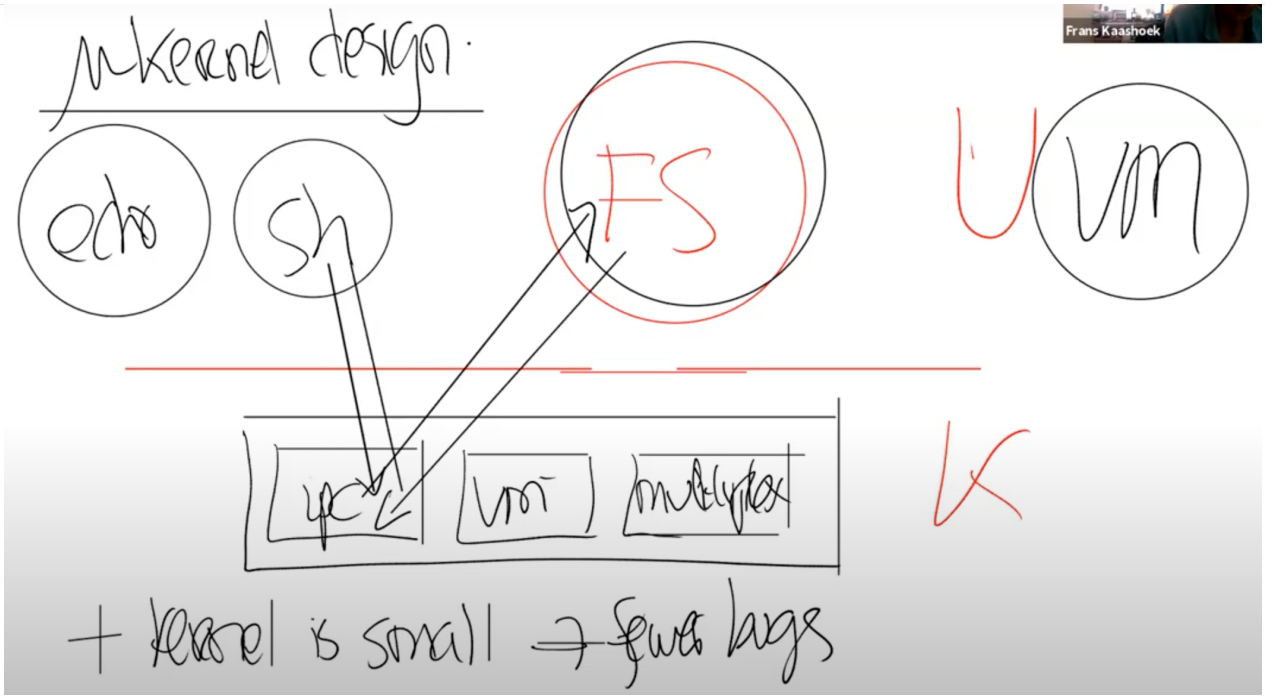


某种程度上来说，这是一种好的设计。因为在内核中的代码的数量较小，更少的代码意味着更少的Bug。

但是这种设计也有相应的问题。假设我们需要让Shell能与文件系统交互，比如Shell调用了exec，必须有种方式可以接入到文件系统中。通常来说，这里工作的方式是，Shell会通过内核中的IPC系统发送一条消息，内核会查看这条消息并发现这是给文件系统的消息，之后内核会把消息发送给文件系统。



文件系统会完成它的工作之后会向IPC系统发送回一条消息说，这是你的exec系统调用的结果，之后IPC系统再将这条消息发送给Shell。



所以，这里是典型的通过消息来实现传统的系统调用。现在，对于任何文件系统的交互，都需要分别完成2次用户空间<->内核空间的跳转。与宏内核对比，在宏内核中如果一个应用程序需要与文件系统交互，只需要完成1次用户空间<->内核空间的跳转，所以微内核的的跳转是宏内核的两倍。通常微内核的挑战在于性能更差，这里有两个方面需要考虑：

1. 在user/kernel mode反复跳转带来的性能损耗。
2. 在一个类似宏内核的紧耦合系统，各个组成部分，例如文件系统和虚拟内存系统，可以很容易的共享page cache。而在微内核中，每个部分之间都很好的隔离开了，这种共享更难实现。进而导致更难在微内核中得到更高的性能。

我们这里介绍的有关宏内核和微内核的区别都特别的笼统。在实际中，两种内核设计都会出现，出于历史原因大部分的桌面操作系统是宏内核，如果你运行需要大量内核计算的应用程序，例如在数据中心服务器上的操作系统，通常也是使用的宏内核，主要的原因是Linux提供了很好的性能。但是很多嵌入式系统，例如Minix，Cell，这些都是微内核设计。这两种设计都很流行，如果你从头开始写一个操作系统，你可能会从一个微内核设计开始。但是一旦你有了类似于Linux这样的宏内核设计，将它重写到一个微内核设计将会是巨大的工作。并且这样重构的动机也不足，因为人们总是想把时间花在实现新功能上，而不是重构他们的内核。

所以这里是操作系统的两种主要设计。如你们所知的，XV6是一种宏内核设计，如大多数经典的Unix系统一样。但是在这个学期的后半部分，我们会讨论更多有关微内核设计的内容。

这里有什么问题吗？因为在课前的邮件提问中，这块问的还挺多的。（没人提问）