1.3 Why Hard and Interesting

我还想说一下为什么我认为学习操作系统是挑战和乐趣并存的？以及为什么我们需要专门针对操作系统设置一门课程？

学习操作系统比较难的一个原因是，内核的编程环境比较困难。当你在编写、修改，扩展内核，或者写一个新的操作系统内核时，你实际上在提供一个基础设施让别人来运行他们的程序。当程序员在写普通的应用程序时，应用程序下面都是操作系统。而当我们在构建操作系统时，在操作系统下面就是硬件了，这些硬件通常会更难处理。在这门课程中，我们会使用一个叫做QEMU的硬件模拟器，来模拟CPU和计算机。这会简单一些，但即使这样，编程环境还是比较恶劣。

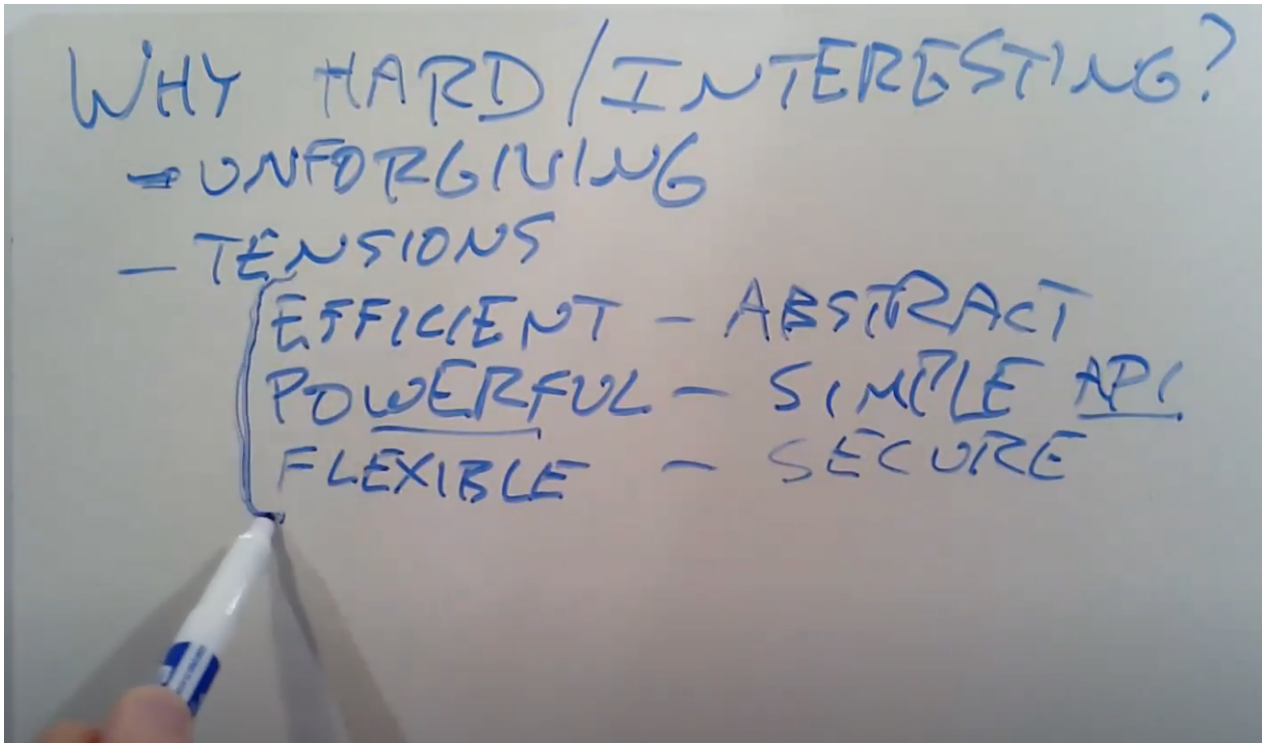
学习操作系统比较难的另一个原因是，当你在设计一个操作系统时，你需要满足一些列矛盾的需求。

- 其中一个是，你想要你的操作系统既高效又易用。高效通常意味着操作系统需要在离硬件近的low-level进行操作，而易用则要求操作系统为应用程序提供抽象的high-level可移植接口。所以，提供一个简单可移植，同时又高效的抽象接口需要一定的技巧。
- 另一个矛盾的点是，我们想要提供一个非常强大的操作系统服务，这样操作系统才能分担运行应用程序的负担，所以我们需要强大的操作系统服务。但同时，我们也想要有简单的接口。我们不想程序员看到数量巨多，复杂且难以理解的的内核接口。因为，如果他们不理解这些接口，他们就会很难使用这些接口。所以，我们也想要简单的API。实际上是有可能提供既简单，同时又包含强大功能的接口。所以，这里要提供一个简单的接口，同时又包含了强大的功

能。

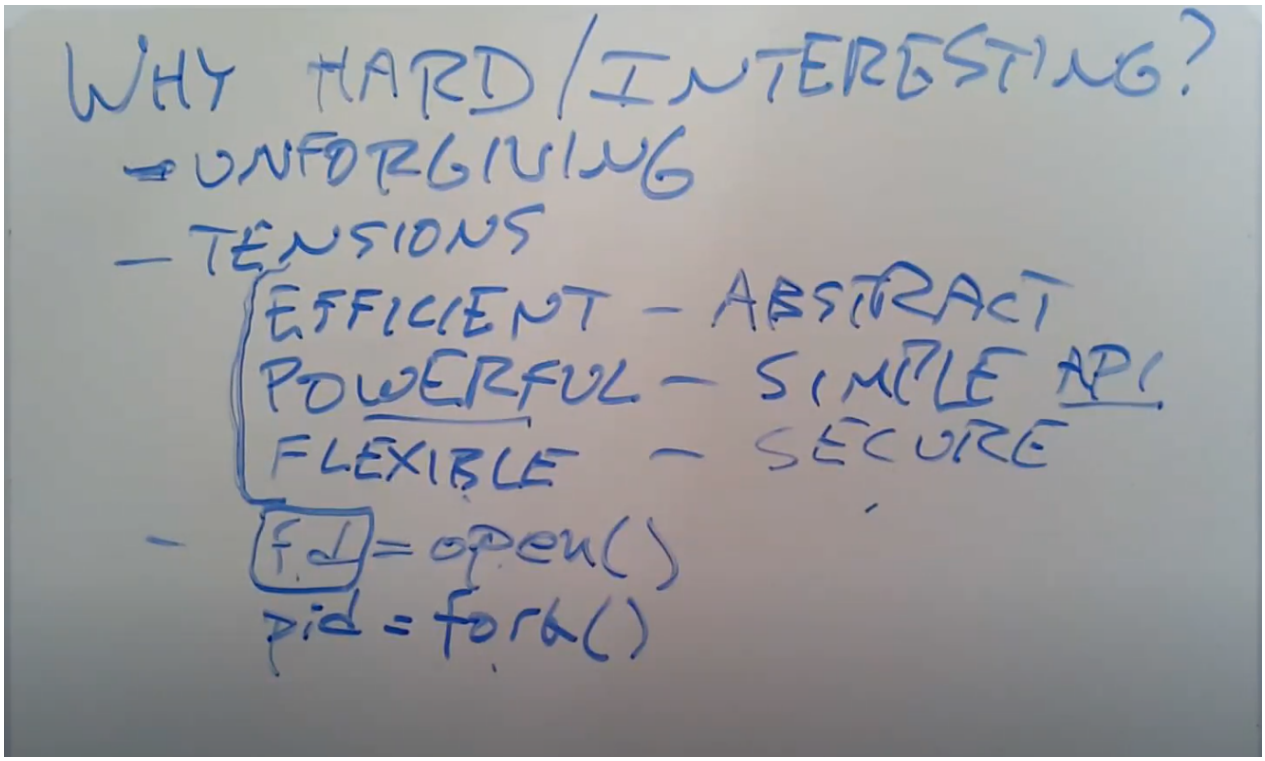
> 学生提问：系统调用跳到内核与标准的函数调用跳到另一个函数相比，区别是什么?
>
> Robert教授：Kernel的代码总是有特殊的权限。当机器启动Kernel时，Kernel会有特殊的权限能直接访问各种各样的硬件，例如磁盘。而普通的用户程序是没有办法直接访问这些硬件的。所以，当你执行一个普通的函数调用时，你所调用的函数并没有对于硬件的特殊权限。然而，如果你触发系统调用到内核中，内核中的具体实现会具有这些特殊的权限，这样就能修改敏感的和被保护的硬件资源，比如访问硬件磁盘。我们之后会介绍更多有关的细节。

- 最后一个矛盾点是所有的操作系统需要满足的。你希望给与应用程序尽可能多的灵活性，你不会想要限制应用程序，所以你需要内核具备灵活的接口。但是另一方面，你的确需要在某种程度上限制应用程序，因为你会想要安全性。我们希望给程序员完全的自由，但是实际上又不能是真正的完全自由，因为我们不想要程序员能直接访问到硬件，干扰到其他的应用程序，或者干扰操作系统的行为。



设计一个好的操作系统还是有可能的，我们后面会大量讨论这个话题。但是想要同时提供上图中两列里面的属性，还是有点难的。

另一件使得操作系统的设计难且有趣的点是：操作系统提供了大量的特性和大量的服务，但是它们趋向于相互交互。有时，这种交互以奇怪的方式进行，并且需要你大量的思考。即使在我之前给出的一个简单例子中，对于open和fork，它们之间也可能有交互。如果一个应用程序通过open系统调用得到了一个文件描述符fd。之后这个应用程序调用了fork系统调用。fork的语义是创建一个当前进程的拷贝进程。而对于一个真正的拷贝进程，父进程中的文件描述符也必须存在且可用。所以在这里，一个通过open获得的文件描述符，与fork以这种有趣的方式进行交互。当然，你需要想明白，子进程是否能够访问到在fork之前创建的文件描述符fd。在我们要研究的操作系统中答案是，Yes，需要能够访问。



另一件有趣的事情，我之前也提到过，操作系统需要能够满足广泛的使用场景。相同的操作系统需要既给数据库服务器使用，又给智能手机使用。随着时间的推移，你的计算机所使用的硬件也在变化，或许你有了超级快的SSD存储而不是机械的硬盘。大概15年前，多核CPU计算机还极其稀有，而现在变得极其的流行。最近，我们又看到了网速以指数级增长。所有的这些都需要时不时的重新思考，操作系统是如何被设计的。

前面理性的分析了一下为什么你要学习这门课程。同时，这里也有一些更加实际的原因来告诉你为什么会选择这门课程。

- 其中一个原因是，如果你对于计算机的运行原理感兴趣，对于你打开计算机以后实际发生的事情感兴趣，那么这门课程你就选对了。
- 类似的，如果你喜欢基础架构，比如你喜欢构建一些其他程序可以使用的服务，那么这门课程都是有关基础架构的内容，因为操作系统就是基础架构。
- 如果你曾经花费了大量的时间来定位应用程序的Bug，或者定位安全的问题，那么你会发现这些经历通常需要理解操作系统是如何运作的。比如从根本上来说，操作系统涉及了很多安全相关的策略。当程序运行出错时，操作系统需要

来收拾残局，而这一步也通常包括在定位问题中。

> 学生提问：对于应用程序开发人员来说，他们会基于一些操作系统做开发，真正的深入理解这些操作系统有多重要？他们需要成为操作系统的专家吗？
>
> Robert教授：你不必成为一个专家。但是如果你花费大量时间来开发，维护并调试应用程序，你最终还是会知道大量操作系统的知识。不论你是否是有意要掌握这些知识，它们就是出现了，而你不得不去理解它们。
>
> 学生提问：对于一些例如Python的高阶编程语言（高阶是指离自然语言更接近，低阶是指离机器语言更接近如C，汇编），它们是直接执行系统调用呢，还是内部对系统调用进行了封装呢？
>
> Robert教授：许多高阶的编程语言都离系统调用较远，这是一个事实。部分原因是很多编程语言想要提供可以在多个操作系统上运行的可移植的环境，所以它们不能依赖特定的系统调用。所以，对于这个问题的答案我认为是，如果你使用了Python，你在某种程度上就与系统调用接口隔离了。当然，在Python内部，最终还是要执行系统调用来完成相应的工作。当然，Python和许多其他的编程语言通常都有方法能直接访问系统调用。