为什么要学习操作系统课程?

Imagination Tech 2014-12-15

几天前，我和同事在喝咖啡时，提起了我将会在秋季给学生上操作系统课程。他的研究领域是不涉及操作系统的，所以他问我这门课的意义何在？学生们应该从这门课里得到什么？这都是一些常规的问题，我想我可以给出合理的答案。我将会在这里详细的阐述一下；如果我遗漏了任何重要的回答，请告诉我。

我上一次教操作系统是在2012年。当时这门课在犹他洲（Utah)的学校是一门必修课并且大约有80个学生在上这门课。现在它不再是必修的了，它已经被归入到一门基于 O'Hallaron and Bryant book的课程。即便如此，现在已经50个学生选了秋季学期的操作系统课，这似乎很合理。这个课程分为两部分：本科生课程和研究生课程。这虽然不是我最喜欢的安排，但也没什么大问题。

也许我应该提一下，一门涉及大量文件系统hacking的OS课程是我对计算机科学感兴趣的原因所在。之前我已经尽量尝试了一些CS课程，但是没有找到太多有挑战性的或者有趣的

用一个简单的案例分析我们我可以推断，操作系统的意义不是去教会学生怎样去编写他们自己的操作系统。第一，这里有些学生对操作系统有兴趣并能编写一个操作系统。他们不需要看课堂上的资料就能够很好地编写自己的操作系统。第二，我们也有一些学生没能力实现一个新的的操作系统或者对实现新的操作系统没兴趣的。他们同样不需要课堂上的资料。

那么，意义何在呢?

并发

编写并发代码并不容易，特别是使用线程共享内存和线程锁。然而，现在很多学习计算机科学的学生都会在他们以后的职业生涯的某些时候使用到（并发）。在OS课程以外的课程里学习并发问题已经成为了一种增长的趋势，但即便如此，操作系统是学生首次了解线程，竞争，死锁等等重要概念的传统课程。教材很难（实际上很简单的，但运用起来很难），在毕业前多看几次是很有帮助的。一个可靠的并发编程介绍对学习操作系统课程是有很大好处的。

资源管理

硬件层次上的资源通常是专用的。操作系统提供了这些资源的种类，它们可以是虚拟的（每个用户都有种错觉，自己拥有资源的一份备份）或者是仲裁的（一次只能有一个用户占有资源，但由操作系统来安排访问顺序）。允许多用户访问专用物理资源是一个很基本的策略并被运用到很多用户级别的程序中。通过详细地学习这些内容，学生学会了能够在许多其他场合重用的模式。

性能分析和冲突解决

正如"为什么#*$是我的机器分页?"。当资源被分享时，冲突通常也会随之而来。冲突问题可以使用多种方式来解决，比如使用队列，合理共享，或者使用优先级。在某些情况下，比如CPU调度，没有单一技术解决方案并且最后的解决方案是一些古怪的混合技术。有时，最令人感兴趣的是找出导致问题出现的主要原因是哪一类冲突。我花费了夏天的一大部分时间去找出所有Windows NT导致MP3跳过的原因。操作系统课是学习这些理论的完美课程，它的适用性比计算机科学更广泛。

隐藏复杂性的接口

一个具有良好设计的接口是一个美妙的东西。它更美妙的地方体现在把一个讨厌的低层次接口（调制解调器或者NE2000卡）转换为一个实用高效的高层次抽象接口（套接字流）。学生应该已经在教材里关于抽象数据类型那部分接触过这些想法了，给定的例子一般都是比较普通的，并且抽象化和隐藏复杂性的作用在那个层次里不够明显。我认为把像套接字（socket），文件系统和地址空间这些集合合到一个单一便利的包里可能是计算机科学10大贡献之一了。这是司空见惯的事，以至于很容易让人忽视它的迷人之处。

没什么神奇的

从用户模式（user mode）看，很容易发现OS是一个神奇的东西–它提供了流畅的多任务处理，高效擦储存管理等。–不好的–它会出现蓝屏，系统颠簸，安全问题和调度异常。对于一般用户来说，这个模式是好的。但在另一方面，如果你想去证明你是一个计算机科学家，你需要知道这些问题的幕后细节。你将会从那里发现什么？很多时候，这看起来都是一些令人忧愁的集合，比如单调的链表，狡猾的启发式资源和维护不当的设备驱动程序。好的OS课程应该教会学生这些：

在内核的代码很优秀，你只需要知道到哪里找到它们。当你第一次看到它们时，你不一定会理解它们。但你理解了它们，你就会学得更多。
通常，内核代码都是很普通的代码。任何人都可以编写它，对比于用户模式代码（user-mode code），内核代码（kernel code）仅需要多一点对细节的关心和注意，因为内核代码中的bug造成的结果更严重。
处理大型软件

这是毫无疑问的，陷入别人的几百万行代码库中去是一个噩梦。错误零散的文档，残旧和广泛的接口，糟糕的交互，和费解的错误信息。不过，欢迎回到现实世界，我们不能因为这些糟糕的问题就经常重新开始。作为一个学生，如果你能够开始制定一个系统的方法去学习你需要用代码修复的大软件的相关部分，那么你以后的生活就会轻松很多。你可以讨厌Linux内核但它比你以后的职业生涯会遇到的软件好多了。

计算机系统设计

设计任何的工程系统，包括软件系统，都是一个权衡的过程。是要侧重于可靠性？性能？消耗还是维护性?因为操作系统是很庞大的，性能至关重要的程序，它一般都要维护几十年，所以它们是学生学习这类权衡的很好的地方。拥有一双发现合适设计点的锐利眼睛的学生在工业上是很需要的。这些人更像一个艺术家而不是一个科学家，你需要看大量的代码，理解这些问题，和学会自己独立考虑这些问题。

总结

我已经尝试去说清楚，一门OS课程不仅仅是关于操作系统和给UNIX/Windows/MacOS的使用者提供知识。优秀的OS课程教会你对广泛使用的操作系统的思考技巧和方式，即使你从没接触过一行的内核代码。实际上，在我的大学里获取学位的CS学生不要求一定要上OS课程，但我觉得，所有真正的计算机科学家要么是已经学这么课，要么已经用其他方式学会了这方面的技巧.

来源：CSDN