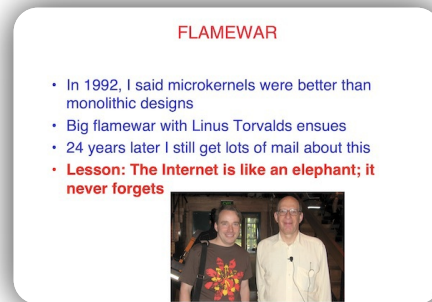


关于微内核的对话

不知怎么的，最近“微内核 vs 宏内核”又成了热门话题。这场争论从 1992 年开始.....



前言

说实话我很久没有关心操作系统了，因为通常所谓的“操作系统”在我心里不过是一个 C 语言的运行时系统（run-time system），就像 JVM 是 Java 的运行时系统一样。由于 C 语言的设计缺陷，这些系统引入了各种无需有的概念（进程，线程，虚拟内存.....），以及它们带来的复杂性和开销。

微内核与宏内核之争当然也包括在内，在我看来这些都是无需有的概念和争论。操作系统相关的领域有很多的“宗教斗争”，比如“Linux vs Windows”，“自由软件 vs 不自由软件”，“RISC vs CISC”，甚至“VIM vs Emacs”..... 人们为了证明自己用的系统或者工具是世界上“最好”的，吵得昏天黑地。遇到有人指出自己用的工具的缺点，随时可能拿枪毙了对方。这些被叫做“flame war”。

我曾经是某些宗教斗争中活跃的一员，不知道这事的人可以去查一下我的历史。等你经历了很多才发现，原来这些宗教情绪和斗争都是那么幼稚无知。

这种“技术宗教情绪”往往显示出参与者心理地位的低下。因

为他们缺乏自信，所以他们的心理需要靠山。这个靠山可能是某种操作系统（比如 Linux），某种编程语言（比如 C++），或者某种工具（比如 VIM）。这些人以自己能熟练使用这些工具为豪，居高临下，看不起“异教徒”。

具有技术宗教情绪的人看似是为了“技术”，“理想”，而其实跟那些以为开着豪车，穿着名牌就是“上流社会”的人是一样低级的，因为他们依靠于这些物品，所以他们的地位在这些物品之下。

一个人需要彻底的把这些东西看成是“东西”，不带有任何崇拜或者鄙视的情绪，他的心理才算是成熟了。

在我的理念里，一个操作系统本应该大概是[这个样子](#)。简单得很，根本不存在那么多问题。我可以利用这些思想来看透现有操作系统的绝大部分思想，管它是微内核还是宏内核。我可以把现有的操作系统看成是这个系统的“退化版”。

操作系统是一个死知识横行的领域。很多人发现操作系统课难学，难理解。里面有些内容，比如各种同步机制，很多人上完课毕了业，工作很多年以后都还弄不明白是怎么回事，它们为什么在那里。类似的东西包括虚拟内存，进程与线程的区别，等等。

经过了很多的经验和思考，加上其他领域给我的启发，我终于明白了。原来很多这些概念都是无须有的，死掉的知识。

操作系统课程里面的概念经常是这样形成的：

1. 很久以前，有人为了解决了一个特定的问题，提出了一个概念（比如 semaphore）。这个概念本来只有一个用途，就是解决他遇到的那个特定的问题。
2. 因为这人太有名，这概念就被写进了教科书里。有时候连他当时的具体实现细节都给写进去了。比如 semaphore 的两个操作被叫做 P 和 V，连这两个名字都给作为“典故”写进去了。
3. 教授们照本宣科，吹毛求疵，要你用这概念解决很多其它问题。很多根本就是人为造出来的变态问题，现实中遇不到的，或者是一些不该用这个概念解决的问题。

这就是为什么操作系统课学起来那么难——很多都是没道理

的难。

再加上 Unix 系统里面一堆设计恶劣，无法有效组合使用的工具软件，操作系统就在学生心中产生了威慑力。死记硬背，喜欢折腾，喜欢发现奇技淫巧的人，在这个领域里茁壮成长。逐渐的，他们产生了莫名的自信。他们并不理解里面的很多概念是怎么来的，只是记住了它们，他们写的代码很难看懂。然后他们开始从心理上打压那些记不住这些概念，看不懂他们代码的人。

久而久之，这些人就成为了大家所崇拜的“神”。

跟有些人聊操作系统是件闹心的事，因为我往往会抛弃一些术语和概念，从零开始讨论。我试图从“计算本质”的出发点来理解这类事物，理解它们的起因，发展，现状和可能的改进。我所关心的往往是“这个事物应该是什么样子”，“它还可以是什么（也许更好的）样子”，而不只是“它现在是什么样子”。不明白我的这一特性，又自恃懂点东西的人，往往会误以为我连基本的术语都不明白。于是天就这样被他们聊死了。

幸运的是我有几个聊得来的朋友，他们不会那么教条主义。于是今天我跟一个朋友在微信上聊起了“微内核 vs 宏内核”这件事。其实这个问题在我脑子里已经比较清楚了，可是通过这些对话，我学到了新的东西。这些东西是我们在对话之前可能都没有完全理解的，也许很多其他人也没理解。所以我觉得可以把这些有价值的对话记录下来。

我不想从头解释这个事，因为你可以从网络上找到“微内核”和“宏内核”的设计原理。我想展示在这里的只是我们的对话，里面有对也有错，翻来覆去的思想斗争。对话是一个很有意思的东西，我觉得比平铺直叙的文章还要有效一些。

对话

好了，现在开始。对话人物“WY”是我，“LD”是我的一个朋友。

（8 月 19 日，开始）

WY: 好多年没折腾 OS, 现在再折腾应该有新的发现。这篇 [paper](#) 说 Minix 3 比 Linux 要慢 510%。

WY: 通常的定义, 说微内核只需要 send 和 receive 两个系统调用。你不觉得有问题吗? 其实函数调用的本质就是 send (参数) 和 receive (返回值), 但只有这两个系统调用, 这种做法是过度的复用 (multiplex) 。

(下载 Minix 3 源代码看了一会儿。上网搜索关于微内核的资料.....)

LD: 是。

LD: 一个外设产生了中断, 中断管理进程接收到中断, 发一个消息给相应的设备驱动进程, 这个进程处理中断请求, 如果设备驱动有 bug, 挂了, 也不会干扰 OS。这就是微内核逻辑。

WY: 微内核似乎一直没解决性能问题。后面的 L4, QNX... 把 sever 隔离在不同的地址空间似乎是个最大的问题。

LD: 导致通讯成本特别大。本来传递个地址就可以的事。现在要整个复制过去。

WY: 地址空间不应该分开。或者也许可以在 MMU 上面做文章, 传递时把那片内存给 map 过去。这样上下文切换又是一个开销..... 函数调用被搞的这么麻烦, 微内核似乎确实是不行。对了, 微内核服务调用时会产生进程切换吗?

LD: 会, 按照微内核的定义, 每一个基本单元都是一个进程。

WY: 完蛋了。

LD: 内存管理是一个进程, IO 管理是一个

进程，每个设备驱动是一个进程，中断管理是一个进程。

WY：进程切换的开销.....

LD：为了降低进程间通信开销，所以定义了 L4。我也不太懂这个有啥用。

WY：改善的是通信开销，但仍然有进程切换开销。我刚看了一下 L4，它是从寄存器传值，但是进程切换会把寄存器都放到内存吧。

LD：对呀，所以 L4 意义似乎不大。

LD：带“微”的除了微软和微信，没一个成功的。

LD：最近流行的所谓微服务。

WY：驱动的 bug 应该有其他办法。

LD：现在的 OS 的问题，就是内核微小的错误，都是让整个系统挂掉。这和我们写软件应该用多进程还是多线程，同样的问题。

WY：应该从硬件底层彻底抛弃现在的进程切换方式。保存的上下文太多。

LD：现在 OS 不是分成 user 和 kernel 保护级别么。我觉得再增加一个两个保护级别，专门针对设备驱动程序似乎是更好的选择。

WY：我以前设想一个办法可以完全不需要

LD：怎么办？编译器静态分析搞

保护级别，而且不需要虚拟内存。

定？ Rust？

WY：完全使用实地址，但是代码无法访问对象外面的内存。

LD：靠编译器保证？

WY：不需要多先进的编译器。语言里面没有指针这东西就行，这样你没法访问不是给你的对象。嗯，需要抛弃 C 语言……

LD： Rust！

WY：还用不着 Rust。其实 JVM 早就是那样了。只不过通常不认为 JVM 是一个操作系统，但操作系统完全可以做成那样。

LD：所谓对象，就是每次地址访问，除了地址还有一个 size？超过 size 不允许？还是编译器确保一定不会超过 size？

WY：你在 Java 或者其它高级语言比如 Python... 都没法访问对象外面的内存啊。只有 C 可以，因为 C 有指针，可以随便指到哪。

LD：是的。C 这种方式，就是天天在没有护栏的桥上走来走去。除了越界访问，还有一个问题，就是多个 task 同时改一块内存。

WY：然后为了防止越界，有了“进程”，“虚拟地址”这种概念。

LD：虚拟地址，还是为了用虚拟内存。

WY: 虚拟地址, 虚拟内存就是为了隔离。每个进程都以为地址从0开始, 然后本来很容易的函数调用被隔离开了。如果改变了这个, 微内核就真的可以很快了。实际上内核就不存在了.....哦, 还是有。就只剩下调度器, 内存管理。IPC 没了, 被函数调用所取代。

LD: 换个思路。其实 OS 最容易出问题的是硬件驱动, 所以尽量让硬件标准化, 别每个硬件都搞一套自己的驱动。让一套驱动支持多种硬件, 问题就解决了。比如 usb 驱动。完全可以做到一类硬件都用一个设备驱动。

WY: 我还是觉得驱动程序 bug 其实可以不导致当机。用内核线程行不行? 共享地址空间, 但是异步执行。

LD: Linux 似乎就是这样。tasklet, 可以被调度的。

WY: 所以驱动程序要是当掉, 可以不死对吗? 我回去查一下。

LD: 看啥错误了。不小心修改了其它模块的内存就完蛋了。其它错误最多硬件本身不能用了。

WY: 所以就是为什么你说再多一个保护级别。

LD: 嗯, 别碰了内核关键的代码。但是驱动之间还是可以互相干扰的。

WY: 是个不错的折中方案。所以微内核解决了一个不是那么关键的问题。

LD: 是的。这个问题不重要。哦, 对了, Windows 是微内核的。好像从 2000 开始。

WY: 只是号称吧。
Mac OS X 不是号称
Mach 微内核加 BSD
吗?

LD: 对。MacOS 也是
微内核。

WY: 那他们怎么解决
的性能问题呢?

LD: 不知道。
Windows 蓝屏可不
少, 显然没做到完全
隔离。至于 Mac, 不
清楚为啥那么稳定。

WY: 根据我们之前的
讨论, Mac 微内核可
能是假的。Mac 稳定
是因为它的 driver 就
没几个吧, 硬件都是
固定选好的。

LD: 嗯, 也是稳定的
主要原因。

WY: 这个英明了.....
而且看来微内核在集
群方面也没什么用
处。

LD: 集群, 每个计算
机是一个 node。挂了
也不怕。

(8 月 20 日继续讨论)

WY: 我发现这个
[paper](#).....

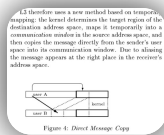
LD: 微内核好处真的
很大么?

WY: 这东西叫
L4Linux, 就是 Linux
跑在 L4 微内核上。比
起纯 Linux, 开销只
有 5%

WY: 代码在这里:
[http://os.inf.tu-
dresden.de/L4/LinuxOnL4](http://os.inf.tu-dresden.de/L4/LinuxOnL4)

WY: L4 的做法是 1)
小参数用寄存器传

递，不切换某些寄存器。2) 大型参数把内存映射到接收进程，跟我之前设想的一样。这样避免了拷贝。然后采用了“direct process switch”，“lazy scheduling”降低了调度开销。现代处理器的 tagged TLB 之类也大大降低了进程切换开销。



WY: 上图是 direct message copy。先把接收进程的地址映射到发送进程的地址空间，然后发送进程往里拷贝。所以其实仍然有一次拷贝，并不像我理想的 OS 那样直接就能传递对象引用，完全不用拷贝。Pass-by-value vs pass-by-reference。但这比起 Linux 似乎开销是一样的。

WY: 好处就是微内核的好处，隔离。可能看各人需求了。一个 99.99% 可靠的系统和一个 99.999999% 可靠的系统的差别？

WY: 不过似乎高可靠需求都去用 vxworks 之类的了

(上网查询 vxworks.....)

WY: 原来 vxworks 也是微内核。

WY: 5% 的开销还可以接受..... 进程切换开销貌似没有提，用的地址映射方法。

LD: cross address space

WY: 刚买了个 tplink 路由器, 里面跑的 vxworks。

LD: tplink 不是 Linux?

WY: 新的 tplink AC1900, 改成了 vxworks。Airport Extreme 也是 vxworks。

LD: why?

WY: 实时, 可靠性高吧。

LD: 可靠性应该是最高的之一。卫星、武器都用。

WY: 波音 787 也用这个, 各种火星车..... 貌似还是说明一些问题。

LD: 我们的一个 mcu 就是 Freescale 的 PowerPC

WY: 还有个 [GreenHills Integrity DO-178B](#) 实时操作系统。F35 用的。

LD: 有个叫“rtems”的 os, 我一直很关注。

WY: Much of the F-35's software is written in C and C++ because of programmer availability; Ada83 code also is reused from the F-22. The Integrity DO-178B real-time operating system (RTOS) from Green Hills Software runs on COTS Freescale PowerPC processors.

WY: Freescale PowerPC...

WY: 摘自 Integrity
DO-178B RTOS:

```
Safe and secure by c
- RTOS designed for
safety critical and
- Based on modern m
- Fast, determinist
```

WY: Integrity 也是微内核。看来微内核是可靠一些，属于在 C 语言框架下的一个不错的折中方案。

.....

(如果你有什么不同意见，欢迎联系我。如果觉得有帮助，可以考虑[付费](#))