微内核操作系统及 L4 概述

杰夫
jliu71@gmail.com

摘要：本文是对微内核操作系统及 L4 的发展历程和主要功能的综述。本文还对微内核操作系统的优缺点及发展前景发表评论。

关键词：微内核，操作系统，L4

Abstract: This paper describes the history of microkernel-based operating systems, and the structure and main functions of L4. It also discusses the pros and cons of microkernel systems and their prospect of actual deployments in the industry.

Keywords: microkernel, operating system, L4

1. Introduction

微内核(microkernel)并非是一个新的概念，这个名词至少在七十年代初就有了。一般认为，他的发明权属于 Hansen [Han70] 和 Wulf [Wul74]. 但是在这一名词出现之前已经有人使用类似的想法设计计算机操作系统了。

早期的操作系统绝大多数是 Monolithic Kernel, 意思是整个操作系统 - 包括 Scheduling （调度）, File system （文件系统）, Networking （网络）, Device driver （设备驱动程序）, Memory management （存储管理）, Paging（存储页面管理） – 都在内核中完成。一直到现在广泛应用的操作系统，如 UNIX，Linux，和 Windows 还大都是 monolithic kernel 操作系统。但随着操作系统变得越来越复杂（现代操作系统的内核有一两百万行 C 程序是很常见的事情），把所有这些功能都放在内核中使设计难度迅速增加。

微内核是一个与 Monolithic Kernel 相反的设计理念。它的目的是使内核缩到最小，把所有可能的功能模块移出内核。理想情况下，内核中仅留下 Address Space Support（地址空间支持），IPC (Inter-Process Communication，进程间通讯)，和 Scheduling（调度），其他功能模块做为用户进程运行。对于内核来说，他们和一般用户进程并无区别。它们与其他用户进程之间的通讯通过 IPC 进行。

在八十年代中期，微内核的概念开始变得非常热门。第一代微内核操作系统的代表作品是 Mach [Mac85]。Mach 是由位于痞子堡的卡内基梅隆大学（CMU）设计。CMU 是美国计算机科学研究重镇，其计算机排名长期位于美国大学前五位。美国只有少数几所大学的计算机是学院不是系，CMU 就是其中之一。除 Mach 外，CMU 的另一重要成果是衡量计算机软件设计能力的 CMM (Capability Maturity Model) 模型，广泛用于评估业界软件公司的计算机软件开发能力。好像印度的软件公司们非常热衷于此，通过 CMM-5 最高规格评价的软件公司们有一半是印度的。

在微内核刚兴起时，学术界普遍认为其优点是显而易见的：

- 支持更加模块化的设计；
- 小的内核更易于更新与维护，bug 会更少。大家知道 Windows 的死机很多是因为 device drivers 造成的。如果把他们移出内核，他们中的 bugs 很可能就不会造成死机；
- 许多模块的 bugs 可被封闭在其模块内，更加易于 debug。软件工程师都知道 kernel debug 是件非常头疼的事情。如果 file system， memory management，和 device drivers 成为一个个独立的进程，不用说这肯定会使 kernel engineers 的日子好过许多。

由于上述原因，很多学术研究人员和软件厂家开始尝试使用微内核的概念设计操作系统。甚至 Microsoft 也有所动作，在设计 Windows NT 时，他们把 UI (User Interface) 从 Windows kernel 中整个拿了出来。由此可看出 microkernel 的流行程度，因为 Microsoft 总是最后一个尝试新想法的。但是这一热潮很快就冷了下来，原因只有一个字：Performance（Well，汉语是两个字：性能）。Microsoft 在 Windows NT 后续版本中，又把部分底层 UI 放回了 Windows kernel。这种现象在计算机界并不少见。在 Java 刚开始流行时，由于其许多 C/C++没有的优点，很多人认为它会很快取代 C/C++成为第一编程语言。但直至今天也没有发生，其中一个重要原因就是 Java 程序的 performance 落后于 C 程序，至今还限制着它在许多场合的应用。

第一代的微内核操作系统的性能，包括 Mach 在内，远不及 monolithic kernel 操作系统。所以大多数人又回到传统技术中去了。Microkernel 也像过时的流行歌曲或减肥方法，很快被遗忘了。

但在九十年代后期，微内核迎来了其生命中的第二春。一些研究人员认真分析了微内核系统性能差的原因，指出其性能差并非根本内在的因素造成，而是设计实现的失误。为证明其论点，他们设计并实现了几个性能远超第一代的微内核操作系统，我们把它们称为第二代微内核系统 [Bar03, Eng95, Lie93A]。其中的一个代表作品就是 L4 [Lie93A, Lie93B, Lie95, Lie96]。

L4 由德国的 GMD 设计。GMD 是德国国家信息技术研究院，相当于中科院计算所加上软件所（但在计算机研究能力方面，GMD 远非中科院可比。与作者一起工作过的几位 GMD 研究人员都是 extremely smart）。L4 的创始人和总设计师是 Jochen Liedtke [Kar05]。此公在 GMD 之后，还供职于 IBM 的 T.J. Watson Research Center，后成为德国 Univ. of Karlsruhe 操作系统方向的正教授（full professor）。了解德国大学系统的人都知道，当德国的正教授可比当美国的正教授要难许多倍，地位也要高许多，因为一个系往往只有一两个正教授。Prof. Liedtke 已于 2001 年过世，但他创建的 L4 正在发展壮大中。近年来，多个运行于不同硬件平台的 L4 系统已被几个不同研究机构设计出来 [SAG03, Tew01]。

本文以后的部分将分析微内核系统的 performance bottleneck 所在，以及 L4 如何试图克服这一困难。微内核系统到底有没有固有的障碍，先天的缺陷？以 L4 为代表的新一代微内核系统的前景如何？请看下文。

2．微内核系统的性能为什么会这麼差？

基于消息传递（Message Passing）IPC 机制是微内核系统的基本特点之一。这一设计理念有助于提高系统的灵活性，模块化，安全性，以及可扩展性。IPC 的性能表现是决定微内核系统性能的关键因素，因为绝大多数系统调用和很多应用程序的服务都需要两个 IPC, 一个服务请求，一个结果返回。

消息传递 IPC 机制现在已经大量用于多种计算机系统中，但是很多 IPC 实现的性能并不太好。根据 CPU 性能和消息长短不同，一个 IPC 大概需要 50 到 500 μs。经过许多研究人员多年的努力，也没有实现 IPC 性能的突破，所以直至最近，消息传递 IPC 被公认为使一个很好的设计理念，但对其适用范围学术界还存在很大争议。

Linux Kernel 创始人 Linus Torvalds 在 2000 年的一段话说得生动（Sorry，是极端），也代表了当时很多人的观点。为了在有些读者受到冒犯时摆脱干系，作者决定不翻译以下这段话。如果您看不懂，good for you.

“Message passing as the fundamental operation of the operating system is just an exercise in computer science masturbation. It may feel good, but you don't actually get anything DONE. Nobody has ever shown that it made sense in the real world.”

由于 IPC 对微内核系统的重要性，其设计人员们也花费了大量时间在改善它的性能上面。微内核系统中 IPC 性能不断提高，但到 90 年代初似乎达到了顶峰。Mach 3 的 IPC 最好性能大约是 115 μs (在 486-DX 50 机器上），其它微内核系统也大致如此。当时许多研究人员认为，有 100 μs 左右的时间是 IPC 的固有消耗，这一时间已无法缩短。但是与之相比，在同样硬件平台上，一个 UNIX 系统调用只需要 20 μs, 好过微内核系统 10 倍。

第一代微内核系统一直未在 IPC 性能上有重大突破，这是导致他们失败的根本原因。对第一代微内核系统的性能分析表明，他的耗时巨大的操作包括用户-内核模式转换，地址空间转换，和内存访问。表面上看起来这一结果似乎合理，但进一步分析发现它们并非真正问题所在。[Lie96] 中给出 Figure 1, 显示这些操作的硬件固

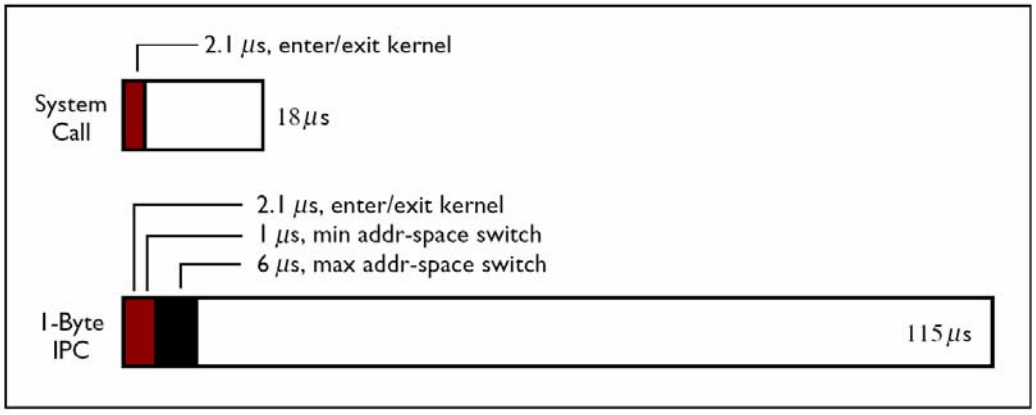

Figure 1. IPC 耗时分析

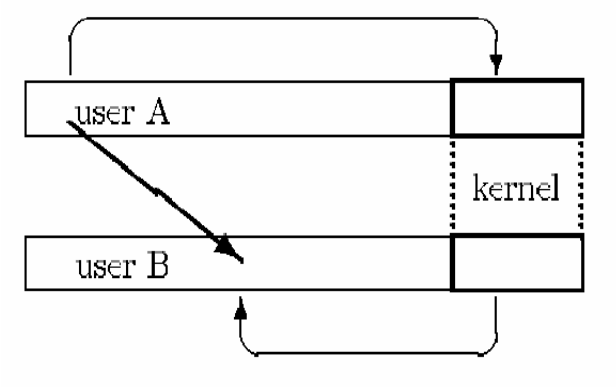

Figure 2. 两次拷贝的信息传递过程

有的时间消耗只占 IPC 总耗时的 3% - 7% （图中深色为硬件固有时间消耗）。

这些证据表明早期微内核的性能差距其实来自于它们的基本构造。早期的微内核系统大多是由 Monolithic Kernel 一步步逐渐改进而来。其很多设计并未发生重大改变。他们虽然被称为微内核，但其代码量还是很大。例如，Mach 3 内核支持 140 个系统调用，代码量为 300K 字节。这种情况不改变，微内核系统的性能恐难提高。第二代微内核系统设计者认识到这一问题，他们对系统内核的基本构造做出重大精简，从而使其性能大步提高。L4 只支持 7 个系统调用，代码量只有 12K 字节。本文下一部分将简单介绍 L4 的一些基本设计，有兴趣的读者可在 [Lie95] 中找到对 L4 的详细介绍。

3. L4 基本结构

L4 是由 GMD 于 1995 年设计，它的两个基本设计原则是：1）高性能和灵活性的要求决定微内核必须尽可能缩到最小，2）微内核实现本身取决于硬件结构，它是不可移植的。虽然微内核可以改善整个系统的移植性，但它本身是不可移植的，因为要达到高性能，它的实现必须紧密联系于硬件结构。

L4 内核支持三种抽象概念：地址空间，线程，和 IPC。他只提供 7 个系统调用，只有 12K 字节代码。在 486-DX50 机器上，一个地址空间切换 IPC, 8 字节参数传递，只需要 5 μs。如果参数量为 512 字节，只需要 18 μs。Mach 3 相应的时间消耗为 115 μs (8 字节) 和 172 μs (512 字节)。

下面我们来看 L4 时如何实现高性能 IPC 的。进程间通信的一个基本问题是数据需要跨越不同的地址空间。大多数操作系统的解决办法是用两次数据拷贝：用户地址空间 A -> 内核地址空间 -> 用户地址空间 B。用户数据被拷贝两次，因为大多数操作系统的地址空间分配模型是用户可防问的地址空间加上内核地址空间，内核空间为所有用户共享。因为用户的地址空间各个不同用户是不同的，所以数据拷贝必须通告内核空间进行，如 Figure 2 所示。

这两次数据拷贝可能耗时很大，如果引起 TLB 和 Cache miss 耗时会更大。如果数据可以由用户空间 A 直接转移到用户空间 B，这将大大提高 IPC 性能。但两个用户直接共享逻辑地址空间会带来一系列的安全问题。L4 的解决办法是通过暂时地址映射：内核把数据的目的地址暂时映射到一个通讯窗口（Communication Window），这一窗口存在于内核地址空间。然后内核把数据从用户 A 地址空间拷贝到通讯窗口供用户 B 使用。如 Figure 3 所示，这一窗口属于内核所有，但用户 B，只有用户 B，可以访问。

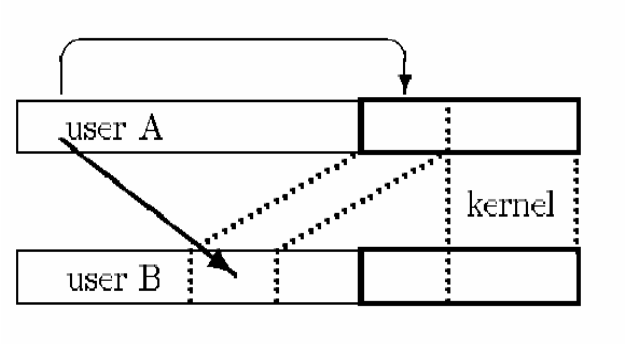

Figure 3. L4 的信息传递过程

除以上讨论的方法之外，L4 还采用了许多新颖的技术来提高内核性能，例如，直接地址转换，懒惰调度（Lazy Scheduling），使用寄存器传送短消息，减少缓存及 TLB Miss 等等。本文不再详述，请参见[Lie95]。

4. Conclusion

微内核操作系统已有二三十年的发展历史，但早期系统的性能不够理想。所以尽管微内核的概念有许多可取之处，但它并未广泛应用于工业界。近年来，新一代的微内核系统，如 L4，Exokernel，在性能上取得了巨大突破，所以学术界，工业界对微内核的兴趣又出现了复苏。对微内核被广泛应用，取代传统 Monolithic Kernel 操作系统，的前景作出预测还为时尚早。但是，至少在一些应用场合，例如，嵌入式系统，实时系统，作者对微内核系统的前景持乐观态度。

References

[Bar03] Xen and the Art of Virtualization, Barham, Paul, etc, ACM Symposium on Operating Systems, Oct, 2003, Bolton Landing, New York.

[Eng95] Exokernel, an operating system architecture of application-level resource management, Engler, Dl, Kaashock, M.F., and O'Toole, J., 15th ACM Symposium on Operating Systems, Dec. 1995, Coper Mountain, Colorado.

[Han70] The mucleus of a multiprogramming system, Hansen, Brinch, Communication of ACM 13, April, 1970, 238-241.

[Kar05] http://i30www.ira.uka.de/aboutus/people/personal/liedtke?lid=en&publ=y, Prof. Liedtke memorial page, Univ. of Karlsruhe, 2005.

[Lie93A] A persistent System in Real Use – Experiences of the First 13 years, Liedtke, Jochen, German National Research Center for Computer Science, 1993.

[Lie94B] Improving IPC by Kernel Design, Liedtke, Jochen, 14th ACM Symposium on Operating Systems, Dec. 1993, Asheville, North Carolina.

[Lie95] On u-Kernel Construction, Liedtke, Jochen, 15th ACM Symposium on Operating Systems, Dec. 1995, Coper Mountain, Colorado.

[Lie96] Toward Real Microkernels, Liedtke, Jochen, Communications of the ACM, Sep., 1996.

[Mac85] http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html, the mach project, CMU, 1985-1994.

[SAG03] The L4Ka:: Pistachio Microkernel, System Architecture Group, University of Karlsruhe, white paper, May, 2003.

[Tew01] VFiasco - Towards a Provably Correct Microkernel, Hendrik Tews, Hermann Härtig, Michael Hohmuth, TU Dresden Technical Report TUD-FI01-1,Jan. 2001.

[Wul74] Hydram: The kernel of a multiprocessing operating system, Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C. and Pollack, F., 1974, Communication of ACM 17, July, 1974, 337-345.