Linux 内存管理实现的分析与研究

肖竟华,陈 岚
(武汉科技大学 计算机科学与技术学院,湖北 武汉 430081)

摘 要 存储管理子系统作为操作系统中最重要的组成部分之一,对整个系统的运行起着举足轻重的作用。Linux 继承了 UNIX 系统的优秀设计思想,并采用了许多先进算法来保持系统的高效性和稳定性。文中先概述了 Linux2.4 物理内存的管理,然后介绍了解决内存中碎片问题的伙伴系统算法和 Slab 分配器,并讨论了它们实现的要点,着重对 Slab 分配器中的几个数据结构进行了分析。

关键词:Linux 内存管理;伙伴系统算法;Slab 分配器
中图分类号:TP333 **文献标识码**:A **文章编号**:1673-629X(2007)02-0187-03

Analysis and Research of Realization for Linux Memory Management

XIAO Jing-hua, CHEN Lan
(Computer School, Wuhan University of Science and Technology, Wuhan 430081, China)

Abstract: As one of the most important parts of the operating system, the memory management subsystem plays a key role in the running of a system. Linux inherited some excellent design ideas from UNIX, and adopted many advanced algorithms to keep the system efficient and stable. This paper summarizes the physical memory management in Linux2.4 kernel. Then it introduces the Buddy system algorithm and Slab allocator that solve the fragment problems in kernel, and discusses the points in realization, especially analyses some main data structures in Slab allocator.

Key words: Linux memory management; Buddy system algorithm; Slab allocator

0 引 言

内存作为最重要的系统资源,其分配和释放策略对系统的运行效率起着至关重要的作用。系统内核和所有进程通过共享有限的物理内存来运行,一个系统的高效性与稳定性往往取决于它内存管理机制。因此,一个高效的内存管理系统不仅要能够有效地管理系统内存,减少频繁分配和回收内存而导致的内存碎片,还要尽量提高分配和回收的速度来提高系统的运行效率。此外,内存管理系统还应该保证内存分配和回收的公平性。

1 Linux 内存管理概述

Linux2.4 支持非一致存储器访问(NUMA)模型[1],根据每个 CPU 对存储器的访问时间,存储器单元被划分为不同的节点(node),每个 CPU 对每个节点访问页所需的时间是相同的。每个节点通过一个类型为 pg_data_t 的描述符来表示。Linux 将每个节点的存储区划分为 3 个管理区(zone):ZONE_DMA,ZONE_NORMAL,ZONE_HIGHMEM。在 80x86 系统上,ZONE_DMA 管理区包含 16MB 以下的页框,可供原有 ISA 设备通过 DMA 直接访问。ZONE_NORMAL 包含大于 16MB 而小于 896MB 的页框。ZNOE_HIGHMEM 包含所有大于 896MB 的页框,不能直接被内核使用。ZONE_DMA 和 ZONE_NORMAL 可以线性地映射到线性地址空间的第 4 个 GB,内核就可以直接进行访问。每个管理区通过 zone_struct 类型结构描述。

Linux 系统用 mem_map 数据结构描述保存系统中所有的页框信息。mem_map 是由 mem_map_t 结构组成的链表,每个 mem_map_t 结构都描述了系统中的一个页框。系统在初始化时,通过调用 free_area_init()函数来初始化 mem_map 数据。

频繁的请求和释放同大小的一组连续页框,必然导致在已分配页框块内分散许多小块的空闲页框,也即所谓的外碎片。另一方面,如果请求存储器的大小与分配给它的大小不匹配,也会造成另外一种问题,即所谓的内碎片。Linux 采用伙伴系统算法和 Slab 分配

收稿日期:2006-05-08
作者简介:肖竟华(1955-),女,湖北武汉人,副教授,硕士生导师,主要研究方向为系统软件开发、操作系统、信息安全。

器模式来解决外碎片和内碎片问题。

2 伙伴系统算法

Linux 系统采用伙伴系统算法管理系统页框的分配和回收,该算法对不同的管理区使用单独的伙伴系统管理。伙伴系统算法把内存中的所有页框按照大小分成 10 组不同大小的页块,每块分别包含 1,2,4,8,…,512 个页框。每种不同的页块都通过一个 free_area_struct 结构体来管理。系统将 10 个 free_area_struct 结构体组成一个 free_area[]数组。在 free_area_struct 包含指向空闲页块链表的指针。此外在每个 free_area_struct 中还包含一个系统空闲页块位图(bitmap),位图中的每一位都用来表示系统按照当前页块大小划分时每个页块的使用情况,同 mem_map 一样,系统在初始化时调用 free_area_init()函数来初始化每个 free_area_struct 中的位图结构。

当向内核请求分配一定数目的页框时,若所请求的页框数目不是 2 的幂次方,则按稍大于此数目的 2 的幂次方在页块链表中查找空闲页块,如果对应的页块链表中没有空闲页块,则在更大的页块链表中查找[2]。当分配的页块中有多余的页框时,伙伴系统将根据多余的页框大小插入到对应的空闲页块链表中。向伙伴系统释放页框时,伙伴系统会将页框插入到对应的页框链表中,并且检查新插入的页框能否和原有的页块组合构成一个更大的页块,如果有两个块的大小相同且这两个块的物理地址连续,则合并成一个新页块并加入到对应的页块链表中,并迭代此过程直到不能合并为止,这样可以极大限度地减少内存的外碎片。

伙伴系统提供了分配和释放页框的函数。get_zeroed_page()用于分配用 0 填充好的页框。和 get_zeroed_page()相似,_get_free_page()用来分配一个新的页框,但是没有被 0 填充,_get_free_pages()用来分配指定数目的页框。通过调用 _free_page()和 _free_pages()可以向伙伴系统释放已申请的页框。alloc_pages_node()是伙伴系统分配页框的核心函数,它有两个变体:alloc_page()和 alloc_pages()。alloc_pages_node()函数在指定的节点中分配一定数目的页框。alloc_page()和 alloc_pages()分别在当前节点中分配一个或指定数目的页框。

3 Slab 分配器

3.1 Slab 分配器基本思想

很多时候,内核需要频繁地申请和释放某一特定类型的对象,如果每次都从伙伴系统中按页框为单位分配和释放内存块,不仅造成大量的内碎片,而且严重影响系统的运行性能。为此,Linux 提供了从缓存中分配内存的机制,即 Slab 分配器。Slab 分配器通过预先分配一块内存区域当作缓冲区,当要求分配对象时就直接从缓冲区中返回,释放对象时 Slab 分配器只是将对象归还到缓冲区以供下次分配时使用,这样就可以避免频繁地调用伙伴系统的申请和释放操作,从而加快申请和释放对象的时间。

3.2 Slab 分配器实现结构

Slab 分配器有 3 层基本结构:缓存(cache)、Slab、对象(object)[3]。其结构可见图 1。

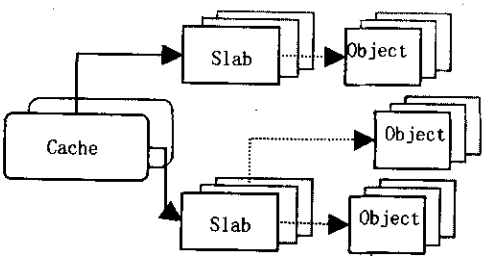

图 1 缓存、Slab 和对象的结构图

3.2.1 缓存

Slab 分配器把对象分组放进高速缓存,每个高速缓存都是同种类型对象的一个集合。缓存通过 kmem_cache_t 结构体来描述,它由多个 Slab 组成。

缓存分为通用和专用 2 种类型。通用缓存由 Slab 分配器自己使用,专用缓存由内核的其他部分使用。Linux 中通用缓存包括两种:

1)一个用于分配高速缓存描述的缓存 cache_cache,此高速缓存在系统初始化时通过 kmem_cache_init()函数创建;

2)两组共 26 个包含几何分布的高速缓存,每组相关的存储区大小分别为 32,64,128,…,131072 字节,一组用于 DMA 分配,另一组用于普通分配。

它们的描述符保存在 cache_sizes[]数组中,在系统初始化时通过 keme_cache_sizes_init()创建。

从缓存中分配一个新的对象,可以调用 kmem_cache_alloc()函数,该函数用于在指定的缓存中申请一个对象。kmalloc()函数也可用从高速缓存中分配一个指定大小的对象,Slab 分配器首先根据要求分配对象的大小在 cache_sizes[]数组描述的通用缓存选择一个通用高速缓存,然后通过 kmem_cache_alloc()从该缓存中分配一个对象实例[4]。kmem_cache_free()用于释放从某个高速缓存中分配的对象。与 kmalloc()对应,kfree()用于释放从通用缓存中分配的对象。

通过调用 kmem_chache_create()函数,可以为一

个特定对象创建一个专用高速缓存。此时，在新创建的缓存中还没有 Slab。只有当第一次要求分配对象时，Slab 分配器才为此高速缓存创建一个新的 Slab。

3.2.2　Slab

Slab 是 Slab 分配器的基本单位，每个 Slab 由一个或多个连续的页框组成。Slab 通过 slab_s 结构体来描述。每个 Slab 中存放一定数量的已初始化的对象，当从指定缓存中申请一个对象时，Slab 分配器从缓存中找一个未满的 Slab，并从中返回一个对象。如果缓存中所有的 Slab 都已满，则 Slab 分配器通过调用 kmem_cache_grow()函数为此缓存添加一个新的 Slab，并从新 Slab 中返回一个对象。Slab 分配器一般不会主动释放 Slab 的页框，即使某个缓存中有空的 Slab，只有当伙伴系统无法满足页框分配请求时，Slab 分配器才会通过调用 kmem_cache_reap()函数从缓存中释放空的 Slab。

Slab 分配器根据对象的大小使用不同的技术来存放 Slab 描述符。当对象比较小时(小于 512kB)，slab_s 存放于 Slab 首部[5]。对比较大的对象，slab_s 位于 cache_sizes 指向的一个普通高速缓存中。

每个对象都有一个类型为 kmem_bufctl_t 的描述符，依次存放在一个数组中，位于相应的 Slab 结构之后，该数组中的描述符与 Slab 中的对象一一对应。

3.3　着色功能

为了充分利用硬件的高速缓存性能，Slab 分配器为每个 Slab 分配一个不同的着色数，即在 Slab 的未用空间上划出一部分空间，放到 Slab 的起始部分，这样将 Slab 中第一个对象分配在相对于该 Slab 起始地址的不同偏移地址(着色数)处，使 Slab 块在硬件缓存区中均匀分布，也使该类对象有良好的地址分布。利用着色功能可以避免 Slab 中对象被映射到相同的高速缓存行上，从而提高了硬件高速缓存的命中率与系统的运行效率。

4　小　结

Linux 内核使用大量的技术来改进内存管理系统，在有效地控制内存碎片的同时提高内存分配和释放效率。在 Linux2.6 内核中，对存储管理子系统进行了一系列的改进，提高了系统的可扩展性，包含对大型服务器如 NUMA 服务器的良好支持，此外还提供了对 MMU 的支持。随着 Linux 内核的不断发展和成熟，Linux 已经越来越具备商业级系统所需的一切品质，完全可以运行非常大的系统来处理科学分析应用程序或者甚至海量数据库。

参考文献：

[1] 厉海燕，李新明. Linux 操作系统特性分析[J]. 微机发展，2002，12(1)：14－18.

[2] Corbet J，Rubini A，Kroah－Hartma G. Linux Device Drivers [M]. 3rd ed. [s.l.]：O'Reilly，2005.

[3] Bovet D P，Cesati M. 深入 Linux 内核[M]. 第 2 版. 陈莉君，冯　锐，牛欣源译. 北京：中国电力出版社，2003.

[4] 楼程辉，孙守迁. Linux 下物理内存管理技术探讨[J]. 计算机应用研究，2000(8)：92－93.

[5] 洪津津，石教英. LINUX 中的 Slab 分配器[J]. 计算机应用研究，2000(11)：82－84.

(上接第 186 页)

好地解决基于不同操作系统和不同编程语言的分布式系统的互操作问题以及遗留系统的集成问题。但随着分布式系统规模的不断增大，对系统的开发周期，以及不同编程语言间，系统组件的可复用性的要求越来越高。文中以“香港利苑集团餐饮管理系统”为背景，使用 MDA 的方法开发分布式系统。在缩短系统开发周期的同时，还有效地解决了不同编程语言间的系统组件的可复用问题。

参考文献：

[1] 汪　芸，顾冠群. CORBA 技术及其应用[M]. 南京：东南大学出版社，1999.

[2] 崔　萌，史耀馨，李宣东，等. 基于 MDA 的 PIM 到 J2EE 平台 PSM 的转换方法[J]. 计算机应用与科学，2005，22(1)：1－2.

[3] Miller J，Mukerji J. MDA Guide Version 1.0.1 [EB/OL]. 2003－06－12. http://www.omg.org/docs/omg/03－06－01.pdf.

[4] 顾峥峰，蔡　勇. 基于 MDA 的 Web 服务开发与集成[J]. 计算机应用，2003，23(5)：142－144.

[5] Huang S，Hu Y，Li C. A CORBA－Based Computer Support Cooperative Work for Dynamic Alliances [J]. Int J Adv Manuf，2002，19：752－755.

[6] 史耀馨，崔　萌，李宣东，等. 基于 MDA 的 UML 模型转换技术——从顺序图到状态图[J]. 计算机工程与应用，2004(13)：40－45.

[7] 沈卓炜，谢俊清，顾冠群. 集成 CORBA 和 UML 的分布式应用开发[J]. 东南大学学报，2001，31：1－5.

[8] Emmerich W. Distributed Commponent Technologies and their Software Engineering Implications [M]. [s.l.]：ACM Press，2002.