操作系统内核内存分配算法的分析与性能评价

张琼声 刘冬萍 （中国石油大学(华东)计算机与通信工程学院 山东东营 257061）

摘要：内存是计算机中的重要资源，快速、合理地分配内存不但能提高内存资源的利用率，也能使系统的整体性能得以提升。本文介绍了操作系统中几种常见的内存分配算法，深入分析了几种操作系统内核内存分配算法（内核分配器 KMA），并对其性能进行了评价。

关键词：内存管理 内存分配 内存释放 KMA 碎片

1 前言

内存储器是计算机中的重要资源，对它们的访问频率和访问方式直接影响到整个系统的性能和效率。如何有效管理、合理分配、使用有限而昂贵的内存资源，对提升系统性能，尤其是内存的使用效率和进程的运行速度显得尤为重要。本文介绍了内存分配的一些基本策略，重点阐述了内存分配中，常见的几种内核内存分配的原理，并对其性能进行了评述。

2 内存分配策略

在内存管理中，操作系统管理所有的物理内存，并为系统内核中的不同子系统以及用户进程分配内存。通常内存的分配策略按分配的内存空间大小可以分为固定尺寸分配策略、可变大小分配策略；按分配内存的地址是否连续可以分为：离散内存分配和连续内存分配策略。

2.1 固定尺寸分配策略

2.1.1 分配策略简介

固定尺寸分配策略的关键在于固定，即当申请内存时，系统总是返回一个固定大小（通常是 2 的指数倍）的内存空间，而不管我们实际需要内存的大小。

2.1.2 固定尺寸分配策略的特点

(1) 优点

延迟时间确定

分配和回收速度快

不会产生太多的细小碎片

(2) 缺点

固定尺寸分配策略最明显的缺点是不灵活。如果定长空间过小，则容易引起溢出，使整个系统在没有任何征兆的情况下崩溃。如果相对保守地配置系统，扩大定长，又会造成内存的浪费，使系统的整体性能下降。

2.2 可变大小分配策略

2.2.1 分配策略简介

可变大小分配策略，即系统处理不同尺寸的分配请求，并分配合适大小的内存空间。

2.2.2 可变分配策略的特点

显然，这样的分配方式很灵活，应用也很广泛，但是它也有自己的缺陷。

(1) 为了满足用户要求和系统的要求，不得不做一些额外的工作（如：分区排序、分区合并等），效率会有所下降；

(2) 在程序运行期间，可能会有频繁的内存分配和释放动作，这样会在内存中形成大量的、不连续的、不能够直接使用的内存碎片，在很多情况下，这对于程序是致命的。

2.3 连续内存分配

2.3.1 分配策略简介

连续内存分配是一种静态的内存分配策略，即在程序开始运行之前就已经决定，每个程序在内存中只占有一块单独的连续内存区域。这种方法因决定的范围不同而发生变化。可以把整个作业作为一个整体进行分配，也可以按每一作业步（即程序）单独分配。

2.3.2 连续内存分配的特点

连续内存分配方法的主要优点是简洁，额外开销也比较低，因为作业被装载在一个连续的区域，在系统运行时不需要内存的分配和回收。但是，连续内存分

配会不同程度的产生内部碎片或者外部碎片。内部碎片是当分配给程序的内存没有被完全利用时出现的不可用内存空间;外部碎片是当一个系统中存在的空闲内存太小至于无法分配给其它程序时出现的不可用的内存空间。浪费了内存资源,对系统的稳定性造成了一定的威胁。

2.4 离散内存分配(分页、分段、段页式)

2.4.1 分配策略简介

离散内存分配是为了解决内存碎片问题而提出的一种分配模式,它允许一个程序占用内存中不连续的区,在程序的执行过程中,通过"地址变换单元"进行地址校正,使程序进行正常的编译和链接。目前常用的分页、分段、段页式内存分配就属于离散型内存分配。

2.4.2 离散内存分配的特点

离散内存分配最大的优点就是有效的避免了内存碎片的产生,提高了内存资源的利用率。但在程序运行过程中,若程序被分配在过多的不连续区域上,就需要有多次的地址校正,这势必降低了程序运行的速度,降低了运行效率。

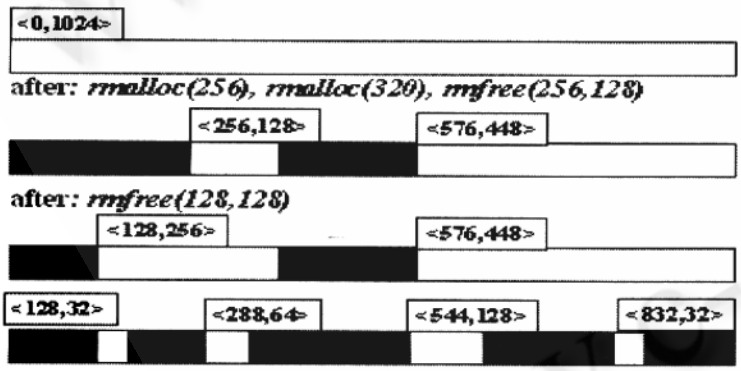

图 1　资源映射图分配器

在现代主流分时操作系统中,大多采用基于分段的页式存储管理,内存分配的基本单位是页。此外,操作系统为内核子系统和用户进程分配内存常采用不同的策略和算法,因为操作系统内核程序所占用的内存块,其使用时间都很短,而且大多数请求所要求的内存大小远小于一个页面(通常为 4KB)的大小,使用页面分配方式显然不合适,因此需要一套独立的机制来实现更细粒度上的内存分配,即内核内存分配。以下就目前比较常用的几种内核内存分配算法(内核内存分配器 KMA)作详细介绍。

3 常用内核内存分配器 KMA 介绍

内核内存分配器管理的内存空间是由页面级分配器为其预留的一部分空间,只用于处理由内核子系统发出的请求。来自用户进程页的请求,交由分页系统处理。因内核内存管理也是对内存进行的操作,也就同样存在效率与碎片问题。

3.1 资源映射图分配器

3.1.1 原理

资源映射图是一系列偶对 < base, size > 的集合。起先,资源映射图只有一个项,即内存空间为空时(如图 1)。当不断的有子系统请求发出后,内存被不断的分配和释放,内存池被逐渐的打碎。资源映射图集合也随着内存的分配和释放增加或者减少。资源映射图按基址顺序组织,这样有利于连接两个相邻的内存区。

利用资源映射图,内核可以采用 3 种策略进行匹配。

最先适合法 从最先满足的请求的空闲块中分配内存。这种方法最快速,也是最常用的分配方法。

最佳适合法 选择可满足内存空间中最小的一块。这种方法容易产生细小的碎片,几乎从来不用。

最差适合法 如果没有正合适的空间块,选择可满足内存空间中最大的一块。比最佳适合法好些,但也很少使用。

图 1 给出了一个管理 1024 字节大小空间的资源映射图。rmalloc 为内存分配操作,rmfree 为内存释放。

从上图我们可以看出,经过若干次的分配释放操作之后,还有 256 字节的内存空间空闲,可是此时内核将无法满足任何大于 128 字节的请求。

3.1.2 性能分析

资源映射图分配器的优点有:算法实现简单,可精确的分配请求所要的内存空间,可部分释放内存,并可完成邻接空间的合并。

然而,资源映射图分配器也存在着显著的缺点:

经过多次的分配释放后,产生了很多的细小碎片,资源浪费率大。

随着内存分片的增加,资源图也在变大。

资源图按照基址顺序组织,虽有利于空间的合并,但

是却增加了内存块分配、回收中,对资源图的线性搜索。

3.1.3 次幂空闲链表分配器

(1) 原理。2 次幂方空闲链表,是把内存按照 2n 划分,其中 n =0 – 9(在 KMA 中通常最小的划分单位为 32KB)。划分后形成大小不等的存储块,同时以一组链表对这些空间块按块的大小分别加以管理,每个块前包含有一个字大小的头结构,块的可用空间不包括这部分(如图 2)。进行内存分配时,通过寻找所需大小的空间块,当分配长度是 2i 大小的块时,从 freelistarr[ ]数组的第 i 条链表开始搜索,找不到再搜索第 i +1 条链表,依次类推。

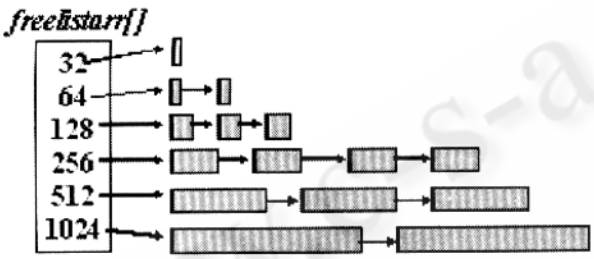

图 2　2 次幂空闲链表分配器

(2) 性能分析。这种方法最大的优点是避免了资源映射图算法中漫长的线性搜索过程,减少了碎片的产生,也使得内存的分配和释放更容易,算法简单,速度相对较快。

但是该算法也存在许多缺点和问题。

每种大小的空闲块分配多少较为合? 太大容易造成内存的浪费,太小则难以满足大的内存请求。

空闲块的划分是以 2 次幂的形式,因此在进行内存分配前需将请求大小取整到 2 次幂,浪费了内存空间,使内存利用率下降。

每个空闲块前的头结构,也使得每个空闲块的实际可用空间小于 2i,当请求大小为 2 的整数次幂时,请求大小 2n 加上头后,分配的内存大小必须为 2n +1,这使得浪费率达到 100% 。

3.2 McKusick – Karels 分配器

3.2.1 原理

McKusick – Karels 分配算法是对 2 次幂分配算法的改进,它需要将被管理的内存组织成一组连续的页,并且同一页面内的所有内存块的大小相同(都是 2 的整次幂)。此外,通过一个使用页数组 kmemsizes[ ]来对页进行管理,每一个页面可以是如下三种状态之一:

空闲 相应的 kmemsizes[ ]中的元素是一个指向下一个空闲页面元素的指针。

被划分为特定大小的内存块 kmemsizes[ ]是内存块大小。

跨页面内存块的一部分 该内存块首页相应的 kmemsizes[ ]元素保存内存块大小。

在 McKusick – Karels 分配器中,因为在同一个页面上的内存块大小相同,分配的内存块不需要头结构保存指向空闲链表的指针,节省了空间,避免了内存浪费,特别是恰好为 2 的整数次幂的分配请求。

3.2.2 性能分析

McKusick – Karels 分配器与 2 次幂分配器有着相同的缺点,此外在出现大量使用页的时候,会出现空闲块到数组之间无限制的移动。但是 McKusick – Karels 分配方法减少了空间的浪费,优化了块的回收与释放。

3.3 伙伴(Buddy)系统

3.3.1 原理

Buddy 系统是克服了固定分配与动态分配之后的一种折衷的分配方法。在并行系统中应用非常有效。它的基本方法是通过不断的对分大内存块来获得小内存块,并尽可能的合并空闲块。当一个内存块被对分后,每一部分称另一部分为自己的伙伴(如图 3)。

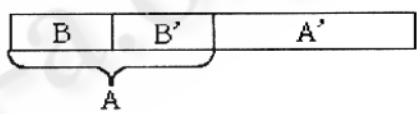

图 3　伙伴系统(其中 A 与 A' 是伙伴,B 与 B' 是伙伴)

3.3.2 实例

在了解了什么是伙伴之后,我们以一个简单的例子解释伙伴算法。这里伙伴系统管理的是一个 1024 字节的内存,最小的分配请求是 32 字节。

(1) 分配(256):将内存对分为 A 和 A'。将 A' 加入到 512 字节的空闲链表中。然后将 A 再对分为 B 和 B',将 B' 加入到 256 字节的空闲链表中,B 返回给客户。

(2) 分配(128):发现 128 的空闲链表为空,检查 256 字节的空闲链表,删除 B'。将 B' 分为 C 和 C',将 C' 加入 128 空闲链表,返回 C 给客户。

(3) 用以上方法分别分配 64 字节和 128 字节的

空间。

(4) 释放(C,128):将 C 加入到 128 字节空闲链表中。此时如图 4 所示。

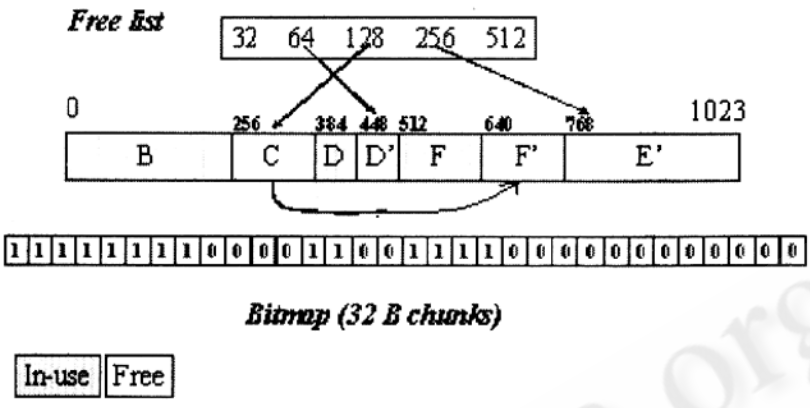

图 4　伙伴系统分配图

在伙伴系统中,请求在分配之前,必须先被取整为 2 的整次幂;在进行分配时,先搜索对应的空闲链表是否有可用的空闲块,有分配器将直接使用它们,没有才进行对分;每一个请求同时更新位图,使它时刻反映内存块的新状态。

3.3.3　性能分析

伙伴系统对内存的划分,使内存可以以不同的大小被利用,提高了灵活性,并且适合于空闲块的合并,对内存的分页系统的改变也较容易。但在块的释放过程中,块的回收过程是一个递归过程,伙伴算法把满足条件的两个块合并为一个块,如果合并后的块还可以跟相邻的块进行合并,那么该算法就继续合并。这会花费大量的时间。另外当分配与释放交替进行时,将可能导致对同一内存块的不断分割和合并。

3.4 Solaris 的 Slab 分配算法

3.4.1　原理

Solaris 的 Slab 分配算法与其它方法相比,具有更好的性能和更高的内存利用。Solaris 的 Slab 分配算法涉及到了对象的状态,更多的关注于对象的重新使用、硬盘 Cache 的利用和分配器 footprint。它为每一种动态分配对象类都分配了对象缓存。当这些对象类需要内存分配时,内核从他们各自的缓存中分配和释放对象。对于其它对象,Slab 通过 kmem_slab 结构对其进行管理。它对 caches 做了有效的管理,增加了从一个 cache 上添加删除存储的机制。提高了资源的利用。

3.4.2　性能分析

Slab 分配器相比于其它方法有显著的提高。它有效的利用了空间,提高了硬件 Cache 和内存总线的利用率。另外它的垃圾收集算法相对简单,提高了碎片的回收速度。Slab 分配器的一个缺点是要为每一类对象都维护一个缓存,增加了管理负担。

4　总结

KMA 的设计涉及到很多问题。它必须能快速、易用、高效的使用内存,使硬件资源得到有效的利用。不同的 KMA 算法各有利弊,使用时需要加以权衡、选择。例如资源映射图分配器可以部分的释放内存,伙伴系统适合于合并内存并与页面系统交互等,在设计操作系统的内存分配算法时应该根据内存资源的数量、对系统关键性能的要求来进行算法的设计和选择。

参考文献

1 UNIX 高级教程系统技术内幕;(美) Uresh Vahalia 著;聊鸿斌、曲广之、王元鹏等译。

2 Bonwick ,J. ,"The Slab Allocator:An Object_Caching Kernel Memory Allocator", Proceedings of the Summer 1994 USENIX Technical Conference,Jun. 1994,pp. 87 – 98.

3 McKusick, M. K. , and Karels, M. J. ,"Efficient Kernel Memory Allocation on Shared – Memory Multiprocessors", Proceedings of the Winter 1993 USENIX Tecchnical Conference,Jan. 1993, 99. 295 – 305.

4 Peterson,J. L. , and Norman, T. A. ,"Buddy Systems",Communications of the ACM,Vol. 20,No. 6, Jun. 1977,pp. 421 – 431.

5 内存碎片处理技术 http://www. guangdongdz. com/special_column/techarticle/jszl19796. html.

6 系统编程与操作系统;(美) D M Dhamdhere 著;徐旭东、金雪云、李昭智等译.