DOI:10.13245/j.hust.2002.07.015

第30卷 第7期 华中科技大学学报(自然科学版) Vol.30 No.7
2002年 7月 J. Huazhong Univ. of Sci. & Tech. (Nature Science Edition) Jul. 2002

一种改进的 Linux 内存分配机制

聂 岚 卢正鼎 董 俊 魏东林

(华中科技大学计算机科学与技术学院)

摘要: 在对 Linux2.4.0 中一种新型的内存分配机制——zone 分配器进行分析的基础上, 针对其中存在的碎片问题提出了一种改进的分配算法和碎片解决方案, 即在区中定义一系列子区, 每一个子区只用来处理一个类对象的分配. 改进后的 zone 分配器简单、快速, 明显减少了内存碎片, 提高了分配性能.

关 键 词: zone 分配器; 区; 子区; 碎片; 对象

中图分类号: TP311.13 文献标识码: A 文章编号: 1671-4512(2002)07-0045-03

1 Zone 分配器

zone 分配器[1] 将内存分为三个区: DMA, Normal, HighMem. 前两个在 Linux2.2 实际上也是由独立的 buddy system 管理的, 但 2.2 中还没有明确的区的概念. DMA 区通常是小于 16 Mbyte 的物理内存区. HighMem 是物理地址超过某个值(通常是 896M) 的高端内存. 其他的是 Normal 区内存, 详见图 1.

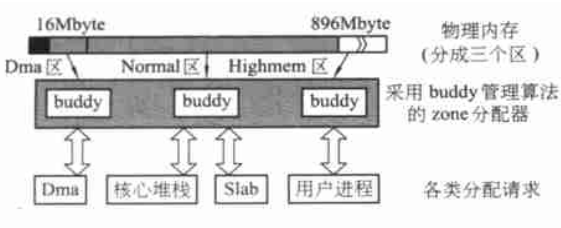

图 1 zone 分配器体系结构

对于任何一类分配请求, zone 分配逻辑为其生成相应的区链表(zonelist), 链表中的每个节点代表可以执行该类对象分配的一个区, 并按照操作优先级排序. 例如, 一个用户页面分配的请求会首先提交到 Normal 区, 若 Normal 区有空闲内存, 则分配给用户, 若没有可用内存, 则这个分配请求会转交到下一个节点 HighMem 区, 最后是 DMA 区, 所以用户页面的分配请求的区链表就是 normal 区→HighMem 区→DMA 区. 再比如 DMA 缓冲区的分配只能在 DMA 区中执行, 所以 DMA 分配请求的区链表就只是 DMA 区一个节点. 表 1 列出了一些常用内存申请的区链表.

表 1 常用内存申请的区链表

Slab, 核心堆栈	用户进程申请的内存	页表	DMA 缓冲区
normal 区 DMA 区	normal 区 HighMem 区 DMA 区	normal 区 HighMem 区 DMA 区	DMA 区

分配时, 分配器找到与分配请求对应的区链表 zonelist, 从链中的第一个区开始, 使用 buddy 算法为其分配合适的空闲内存, 若找到, 则成功返回; 若没找到, 则在下一个区中分配, 如此类推. 回收时, 从指定的页面地址处释放内存块, 调用 buddy 算法进行合并.

2 一种改进的分配机制和碎片解决方案

Linux 系统中, 大部分申请都是大小为一个页面的对象(小对象), 但是, 也存在一些对大对象的申请, 例如进程的核心堆栈(8 Kbyte), 还有来自 SCSI 和网络层的分配请求(例如一个超过 16 Kbyte的包) 等. 大对象一般有 2^n 个连续页面 ($n \geqslant 1$), 而且对实时性要求很高. 页面碎片的存在会造成这些大对象的申请频繁失败, 由于这些请求通常发生在底半层和中断子程序中, 如果分配失败则很难控制, 会影响系统性能和稳定性.

buddy 算法通过释放内存时合并相邻空闲块可以一定程度地减少碎片, 但是由于不同类型和大小的对象混杂分布, 各种对象的生存期不同, 再

收稿日期: 2001-11-09.

作者简介: 聂 岚(1977-), 女, 硕士研究生; 武汉, 华中科技大学计算机科学与技术学院 (430074).

基金项目: 国防科技预研基金资助项目.

加上分配请求的随机性, 运行了一段时间之后, 因此也很难能合并出连续的数个页面来满足大对象请求, 而且合并的开销太大. 针对上述问题, 本文提出了一种改进的分配机制和碎片解决方案, 在区内用面向对象的分配算法来取代 buddy 算法, 同时在后台执行碎片清理程序. 通过分析和测试, 减少了页面碎片, 提高了分配效率.

2.1 结构

如前介绍, 目前 Linux 系统的物理内存被分为 3 个区, 现在把这个分区的思想扩展一下. 在每个区中划分若干个子区(subzone), 每个子区一般情况下只用来处理一类对象的分配, 但在最坏情况下, 由于分配需要, 也可能要处理个别其他类型的对象的分配, 这些对象称为临时对象. 子区结构如图 2 所示. 一般情况下, 子区被等分成对象大小相同的块, 空闲对象之间用指针链接.

图 2 子区结构

subzone– struct 结构用来管理一个子区, 包括子区大小 size, 类型 type(在 normal 区中的子区类型均为 normal, 以此类推), 指向空闲对象链表的指针 free– object, 子区的物理首地址 start– addr 等. 同时, 结构中还留有一些统计信息, 如子区内已分配对象个数 cnum– active, 变量 s– flag 来标识子区内是否存在临时对象.

在新机制中, 对象和子区之间的匹配关系是由一个二元映射表来维护的. 在每一个区中, 都有一个这样的表, 入口是对象类型, 出口是该区中可分配该类对象的所有子区所组成的链表. 这是由于用来分配某类对象的子区可能不止一个, 它们组成一个半有序的链表, 先是空子区(子区中所有对象都已被分配), 接着是半空子区(部分对象被分配). 子区链由管理结构 subzonelist– s 来管理, 其中, 成员变量 free– zone 指向链中首个半空子区. 表的最后一项存放区中所有空闲子区(子区内未分配任何对象)组成的链表.

2.2 分配和回收

二元映射表是在分配的过程中逐渐建立起来的. 在未执行任何分配之前, 每个子区都是空闲子区. 有分配请求时, 分配逻辑找一个空闲子区来执行分配, 此后, 这个子区只能用来处理此类对象的分配, 将这种对应关系添加到映射表中. 之后, 按下列算法进行分配

a. 当进程提出一个分配请求时, 分配逻辑为其找到相应的区链表, 并将该请求首先提交到链表的第一个区中.

b. 分配请求的 gfp– mask(用来描述分配请求的标志符)标志, 对象大小(order)和对象 id 构成一个组合关键字, 通过这个关键字查找映射表, 找到相应的子区链, 通过 free– zone 指针, 找到第一个半空子区, 搜索 free– object 找到子区中首个空闲对象, 分配给申请者, 成功返回. 如果所有的子区中对象都已被分配, 则进入下一步.

c. 找一个空闲的子区(映射表的最后一项), 将其加入到映射表中用来分配此类对象的子区链的尾部. 把子区等分成对象大小相同的若干块, 将第一块分配给申请者, 成功返回. 若未找到空闲子区, 则进入 **d**.

d. 将请求提交到区链表的下一个区中, 重复 **b** 和 **c** 操作. 如果仍然没有找到空闲对象, 也就是最坏的情况: 三个区中的空闲子区个数都为 0, 而且与对象匹配的子区链表中空闲对象的个数也为 0. 进入 **e.**

e. 重新回到区链表的第一个区, 搜索区中的每一个子区, 直到找到指定大小的空闲块. 虽然找到的空闲块并不在与对象匹配的子区中, 也分配给申请者, 称为临时对象. 标志该子区为含有临时对象的子区.

回收算法很简单, 释放对象时将其插入子区中的空闲对象表, 将 cnum– active 减 1. 在一个子区内所有的对象都被释放, 也就是子区的引用记数 cnum– active 为 0 时, 将子区从相应的链中移走, 加入到空闲子区链表中.

2.3 回收和碎片清理

为了不导致最坏情况下某些操作的异常行为, 应该在后台进行适当的碎片清理. 当系统监测到区中空闲子区的个数小于某个下限时, 而且各类子区链表中的空闲对象个数严重失调时, 会在后台启动碎片清理程序.

碎片清理程序对那些对象分配松散的子区进行操作, 如图 3 所示, 这是一个缓冲区页面的子区链, 一段时间后, 各个子区都有了一些空闲对象. 碎片整理程序将子区 3 中的两个活动对象移动到区 1 或区 2 的空闲对象的位置上, 从而可以腾出一个空闲的子区为那些对象分配紧张的请求服务. 在最坏情况发生之后, 系统应监视资源的使用情况, 一有机会(区中存在空闲子区, 或是与临时对象匹配的子区中有空闲对象时), 就将临时对象从当前所在的不匹配的子区中移动到匹配的子区中, 从而尽可能地保证每个子区中只含有一类对

象.

图 3　可执行碎片清理的子区
(白色代表空闲对象,黑色为活动对象)

2.4　性能分析

a. 减少了内存碎片

按照这种分配方法,同类的对象紧密地排放在一起,减少了碎片剩余;同一个子区中的对象由于具有同样的生存期,因此一次性可以释放一个子区大小的连续空间,释放的空白子区可供其他的对象分配任务使用,提供了一种内存复用的机制;如果按原分配算法,这些对象零散的分布在物理内存中,与不同类型,生存期不同的对象混杂在一起.对象释放时,邻居可能还处于活动状态,所以不能合并出大块的空闲内存,反而只会在物理内存中形成更多孤立无用的零星碎片,碎片的累积会影响系统性能.

b. 分配回收简单,属于常数时间的操作

这种新的面向对象分配算法的特点是为每一种类型的对象维护了一条可提供分配的子区链表.分配时,只需通过查表,找到相应的子区链,通过双层 free 指针索引就找到合适的空闲块,同时修改引用计数;回收也只是将空闲对象列表的简单操作,避免了 buddy 系统中复杂的合并算法所带来的开销.

c. 提高了系统性能

buddy 系统通过不断地将内存对分或合并,以此来适应内存请求的变化,每释放一次内存块,分配器都尽可能进行合并.当分配与释放交替进行时,算法可能刚合并完内存块,马上又要把它分裂了.合并的过程是递归进行的,这导致最坏情况下极差的性能.而改进后的Zone 分配器采用双层分区的思想,为每一种动态分配的对象类提供空闲对象池(也就是子区),不同类对象的分配回收相互隔离,互不干扰,仅是对子区中空闲列表的简单操作,不存在上述的问题,因此系统性能有了较大的改善.

d. 对内存压力的动态变化感应灵敏,可扩展性好

系统通过监测各子区的一些统计信息,能灵敏的感应内存压力的动态变化,进行适当的碎片清理.

为了测试改进后的分配机制处理碎片和分配效率的性能,本研究编制了一测试脚本,分别从总碎片,平均分配回收时间和每 s 执行脚本数目三方面来进行统计.在奔腾 4(256 Mbyte 内存,40 Gbyte Maxtor 硬盘)分别运行现有的 Linux2.4.0 版本和修改后的版本,通过一个测试脚本对两种不同分配机制下的内存碎片,分配效率和每 s 执行脚本数目进行统计,得到了表 2 中的实验数据.由表 2 可知,改进后的 zone 分配器简单,快速,明显减少了内存碎片,提高了分配性能.

表 2　性能比较表

性能标准	改进前	改进后
总碎片(浪费)比例/%	46	35
平均分配+释放时间/μs	9.4	4.1
每 s 执行的脚本的数目	199	208

参　考　文　献

[1] 庞丽萍. 操作系统原理. 武汉: 华中理工大学出版社, 1994.

An improved mechanism for memory allocation in Linux

Nie Lan　Lu Zhending　Dong Jun　Wei Donglin

Abstract: The performance analysis of the zone based on buddy system, a new mechanism for page-level memory allocation in Linux2.4.0 was made. The memory fragmentation was described, and an improved mechanism to lessen fragmentations in which every zone was divided into a number of sub-zones and each sub-zone was used for one type of allocation was presented. The test result demonstrated that the mechanism lessened the fragmentation and enhanced performance.

Key words: zone allocator; zone; sub-zone; fragmentation; object

Nie Lan　Postgraduate; College of Computer Sci. & Tech., Huazhong Univ. of Sci. & Tech., Wuhan 430074, China.