对linux伙伴系统及其反碎片机制的研究

雷方杰
（广东工业大学计算机学院，广东 广州 510006）

摘要：首先介绍了linux伙伴系统的原理，然后提出了旧版本内核在解决碎片问题上的不足，并详细分析了内核对于该问题的最新解决方法。

关键词：内存管理；linux伙伴系统；外碎片
中图分类号：TP316.81　　**文献标识码**：A　　**文章编号**：1673-1131 (2011) 06-0075-02

伙伴系统是一种经典的内存管理方法。Linux伙伴系统的引入为内核提供了一种用于分配一组连续的页而建立的一种高效的分配策略，并有效的解决了外碎片问题。

1 linux的内存组织

在linux中，内存被划分为多个结点，每个结点与系统中的某个处理器对应。每个节点又被划分为最多三个内存域：ZONE_DMA，ZONE_NOMAL，ZONE_HIGHMEM。ZONE_DMA是用于DMA操作的内存区。ZONE_NOMAL是可以直接映射到内核段得普通内存区。ZONE_HIGHMEM为高端内存区。每个内存域把该内存域中的空闲块分组为MAX_ORDER个块链表，每个链表中包含着许多同样大小的连续内存块。对于第order个链表，该链表管理着所有大小为2^{order}的空闲块，如图1所示。

2 伙伴系统的结构组织

系统的每个内存域描述结构如下：

```
struct zone {
……
struct  free_area    free_area[MAX_ORDER];
……
};
```

free_area数组定义如下：

```
struct free_area {
struct list_head  free_list[MIGRATE_TYPES];
unsigned long        nr_free;
};
```

nr_free指定了当前内存域中的空闲内存块的数目。每个空闲内存块包含一个或多个连续页。阶是伙伴系统用来描述内存分配的数量单位。数组free_area[MAX_ORDER]管理其所在内存域的所有空闲块。如图1所示，数组free_area[MAX_ORDER]下标解释为阶，该数组的每一项free_area[order]都指向了一个双向链表,该链表将所有大小为2^{order}个页的连续空闲内存块链接在一起。

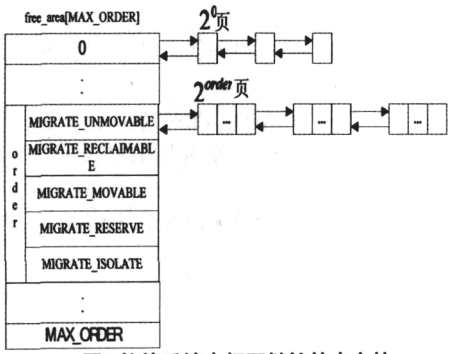

图1 伙伴系统中相互链接的内存块

3 伙伴系统内存分配与释放

阶是伙伴系统用来描述内存分配的数量单位,所以伙伴系统总是选择2^{order}个连续页用来分配，order为0-MAX_ORDER的整数。

3.1 内存的分配

如果需要分配1页的内存时，内核会从内存域zone的当前处理器的每CPU高速缓存中申请。如果每CPU高速缓存结构热页链表pcp[0].list中的页数pcp[0].count不为零,则会从链表头部选择一页用来配。如果pcp[0].count等于零，则free_area[0]中移除pcp[0].batch个页，并将这些页填充到每CPU高速缓存结构热页链表pcp[0].list中,然后从链表头部选择一页用来分配。

如果需要分配多页，这里分两种情况说明。第一，当分配页数满足2^n形式时，可从伙伴系统的n阶空闲列表free_area[n]中选择一块用来分配。如果free_area[n]中没有满足条件的空闲块，则从较高的分配阶分配一块内存，将按照伙伴系统原理将该内存块分裂成两块相等的内存块，将其中的一块用来分配，以此类推。假设n=2，在free_area[2]和free_area[3]中没有找到空闲的可用块，而从free_area[4]中找到了可用的空闲块,伙伴系统拆分内存块的步骤如下：(1)将free_area[4]中选择的空闲块从该链表中移除。(2)内核切换到低一个分配阶3，先将步骤1中的空闲块拆分为两半，将后一半插入到阶为3的空闲列表中。前一半用来分配，由于前一半内存块的大小为2^3页，仍然大于要分配的2^2页。因此需要对该内存块进行进一步的拆分。(3)将该内存块拆分为两块大小为2^2页的内存块，后一半插入到阶为2的空闲列表中，前一半则用来分配。第二，当需要分配的页数pages不满足2^n形式但满足$2^{i-1}<pages<2^i$,内核会分配2^i个连续页而非pages页的内存块。分配原理和第一种情况一样。

3.2 内存的释放

内核对单页和多页内存块的释放采用不同的方式。当释放单页的内存时，内核将其置于每CPU高速缓存中，对很可能出现在cache的页，则放到热页的列表中。内核先判断每CPU高速缓存中的页数pcp->count是否大于或等于pcp->high,如果是，则将数目为pcp->batch的一批内存页还给伙伴系统，然后将该页添加到每CPU高速缓存中。

当释放多页的块时，内核首先计算出该内存块的伙伴的地址。内核将满足以下条件的两个块称为伙伴：(1)两个块具有相同的大小，记作b。(2)它们的物理地址是连续的。(3)第一块的第一个页框的物理地址是$2\times b\times 2^{12}$的倍数。

如果找到了该内存块的伙伴，确保该伙伴的所有页都是空闲的，以便进行合并。例如：两个大小为都为2页的伙伴合并成一个2^2页的内存块。内存继续查找该2^2页内存块的伙伴并检查

其是否空闲，如果空闲则继续将这对伙伴合并为大小为2^3的空闲块。重复该合并过程直至其伙伴不处于空闲状态，假设此时块大小为2^4,内核将该块放置到伙伴系统4阶的空闲链表free_area[4]中。

4 伙伴系统反碎片机制

4.1 旧版本内核在内存管理方面问题

Linux在内存管理方面一直存在一个问题：在系统长期运行后，物理内存会存在大量的碎片。在大部分内存仍然尚未分配时，也可能发生碎片问题。例如图2，在该32页连续的内存空间中，其中有4页已经被分配，但可分配的最大连续区只有8页。linux 新的反碎片机制试图从最初开始尽可能的防止碎片，该机制的工作原理见下文。

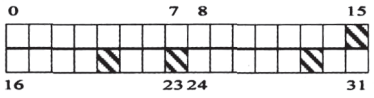

图2 少量的页导致大量的碎片

4.2 反碎片机制的工作原理

内核将已分配页分为以下三种不同的类型：(1)不可移动页：这些页在内存中有固定的位置，不能够移动。(2)可回收页：这些页不能移动，但可以删除。内核在回收页占据了太多的内存时或者内存短缺时进行页面回收。(3)可移动页：这些页可以任意移动，用户空间应用程序使用的页都属于该类别。它们是通过页表映射的。当它们移动到新的位置，页表项也会相应的更新。

内核的反碎片正是基于将相同类型的页分组的思想。这样，对于图2，假设分配的页都是不可移动的，而多数空闲页是属于可回收的。如果将这些页分别放到不同的列表中，如图3所示，这样在可回收页列表中可以剩余足够大的连续空闲页用于分配。

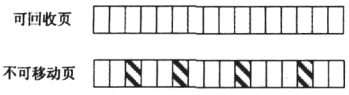

图3 将页分组，从而减少碎片

因此，内核为了便于将不同类型的页分组定义了以下6个宏用于表示不同的迁移类型，分别使MIGRATE_UNMOVABLE，MIGRATE_RECLAIMABLE，MIGRATE_MOVABLE，MIGRATE_RESERVE，MIGRATE_ISOLATE，MIGRATE_TYPES。当内核在前三个迁移列表中无法满足分配条件时，会从紧急列表MIGRATE_RESERVE中来选择页用于分配。MIGRATE_ISOLATE用于跨越NUMA结点移动物理内存页，它有益于将物理内存页移动到接近与使用该页最频繁的CPU。MIGRATE_TYPES表示迁移类型的数目，即上述五种迁移类型。由上文free_area的定义可知内核将每阶空闲内存区又分为MIGRATE_TYPES(5)个链表，如图1所示，第order阶有五个链表：他们分别order阶不可移动空闲内存块链，order阶可回收空闲内存块链表，order阶可移动空闲内存块链，order阶紧急分配空闲内存块链表，order阶跨越NUMA结点的可移动空闲内存块链表。

内核分配器在分配内存时通常需要指定要分配页的迁移类型，此时内核会从指定阶order的空闲区free_area[order]中指定的迁移列表中去查找适当的连续内存块用于分配，如果无法找到适当的连续内存块，则内核会按照分配阶从大到小遍历free_area[MAX_ORDER-1] ~ free_area[order]的备用的迁移列表中的空闲内存块，直到找到合适的内存块。内核定义了一个二维数组fallbacks用来指定当指定的迁移类型的空闲列表耗尽时，该在哪个备用的迁移列表中查找分配。

在内存子系统初始化期间，所有的页都被标记为可移动的。在启动期间，核心内核分配的内存是不可移动的。此时内核的策略是分配一个尽可能大的连续内存块，将其从可移动列表转换到不可移动列表。分配一个尽可能大的连续内存块而不是选择更小的满足要求的内存块的原因如下：

例如：现有一块可移动的具有32页的连续空闲内存块。当内核需要分配1页不可移动内存页时。分配器选择后16页(而不是一页)转移到不可移动列表，然后从不可移动列表中选择1页（例如17）用于分配，如图3所示。如果内核又需要分配1页不可移动内存页则从不可移动列表中选择一页用于分配，图3显示了进行4次分配后内存的分布。

但如果内核只选择1页用于分配将其加入到不可移动列表，当下次需要分配时又选择一页加入到不可移动列表，图4显示了进行4次分配后内存的分布。

图4 分配一个尽可能大的连续内存块

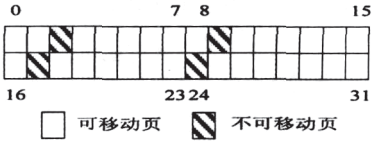

图5 分配小的内存块

因此，当没有满足可用于分配的不可移动空闲块时，分配器会在可移动列表中迁移一个尽可能大的连续内存块给不可移动列表。这样避免了启动期间内核分配的内存散列到物理内存各处，从而使其他类型的内存分配免受碎片的干扰。

5 结语

伙伴系统是基于一种简单而强大的算法。然而碎片问题是linux内存管理一直存在的一个问题。linux内核的反碎片机制只是从最初开始时尽可能的防止碎片，而不能完全禁止碎片的产生。

参考文献:

[1][德]Wolfgang Mauerer,郭旭译.深入Linux内核架构[M].北京：人民邮电出版社,2010.

[2][爱尔兰]MEL GORMAN,白洛,李俊奎,刘森林,译.深入理解Linux虚拟内存管理[M].北京:北京航空航天大学出社,2006.

[3]毛德操,胡希明.Linux内核源代码情景分析(上册)[M].杭州:浙江大学出版社,2001.

[4]DANIEL.P BOVET&MARCO CESATI,陈莉君,张琼声,张宏伟,译.深入理解linux内核[M].北京:中国电力出版社,2007.

作者简介:雷方杰(1987-)，湖北人，男，硕士研究生，研究方向为嵌入式系统与智能技术。