DOI:10.13682/j.issn.2095-6533.2011.04.031

Linux 内核中内存池的实现及应用

王小银,陈莉君

(西安邮电学院 计算机学院, 陕西 西安 710121)

摘要:基于对 Linux 操作系统内存管理机制、算法和模式的分析,详细解读 Linux 内核 2.6 版本中有关内存池的定义内涵,明确内存池的创建方法和调用原理,并给出内存池在网络文件系统中的应用。实验表明,在内存实现及用户程序中使用内存池进行内存管理,可以减少内存碎片,提高分配速度,防止内存泄露。

关键词:Linux 内核;内存池;内存管理

中图分类号:TP316.1 文献标识码:A 文章编号:1007-3264(2011)04-0040-04

存储器是一种宝贵而又紧俏的资源,如何对它加以有效的管理,不仅直接影响到存储器的利用率,而且还对系统性能有重大影响[1]。在 Linux 操作系统中,内核将物理页作为内存管理的基本单位,内存管理单元通常以页为单位进行处理。通常我们习惯直接使用 new、malloc 等 API 函数申请分配内存。但在申请内存时,由于所申请内存块的大小不定,当频繁使用这些 API 函数时会造成大量的内存碎片从而降低性能[2]。因此 Linux2.6 内核引入了一种更有效的内存分配方式——内存池(Memory Pool)。但目前用户的应用程序仍多用默认的内存管理函数 new/delete 或 malloc/free 分配和释放内存,这样会有一些额外的开销,也有可能导致内存泄露。本文对 Linux 操作系统内核管理方法进行分析,研究 Linux 内核 2.6 版本中内存池定义、创建、实现及其在网络文件系统中的应用,从而用户可以在应用程序中采取内存池的分配与释放方法进行内存管理,提高内存的分配速度,防止内存泄露。

1 Linux 系统内存管理

在 Linux 内存管理中,简化了分段机制,使得虚地址与线性地址总是一致的,线性空间在 32 位平台上为 4GB 的固定大小。Linux 内核这 4GB 的空间分为两部分,其中 0 至 3GB 是用户态空间,由各进程独占;3GB 到 4GB 是内核态空间,由所有进程共享,但只有内核态进程才能访问[3]。

Linux 内存管理采用了伙伴算法,伙伴算法分配内存时,每次至少分配一个页面。这种方法适合大块内存请求,不适合小内存区请求。当请求分配的内存大小为几十个字节时需要例外的管理机制。从 Linux2.2 开始,内存分配时采用了一种空间和时间上都具有高效性的内存分配器——slab 分配器[4]。

图 1 是 slab 分配器的主要结构。最高层 cache_chain 是一个 slab 缓存的链接列表,这对于最佳适配算法非常有用,可以用来查找大小最适合的缓存(遍历列表)。cache_chain 的每个元素都是一个 kmem_cache 结构的引用(称为一个 cache)。它定义了一个要管理的给定大小的对象池。

与传统的内存管理模式相比,slab 缓存分配器有很多优点。首先,内核通常依赖于对小对象的分配,它们会在系统生命周期内进行无数次分配。slab 缓存分配器通过对类似大小的对象进行缓存而提供这种功能,从而避免了常见的碎片问题。其次,slab 分配器还支持通用对象的初始化,从而避免了为同一目标而对一个对象重复进行初始化。最后,

收稿日期:2011-03-28
基金项目:西安邮电学院中青年科研基金资助项目(103-0439)
作者简介:王小银(1976-),女,副教授,硕士,研究方向:操作系统安全、嵌入式系统,E-mail:wangxiaoyinxy@126.com;
陈莉君(1964-),女,教授,硕士,研究方向:Linux 操作系统、嵌入式系统。

slab 分配器还可以支持硬件缓存对齐，这允许不同缓存中的对象占用相同的缓存行，从而提高缓存的利用率并获得更好的性能。

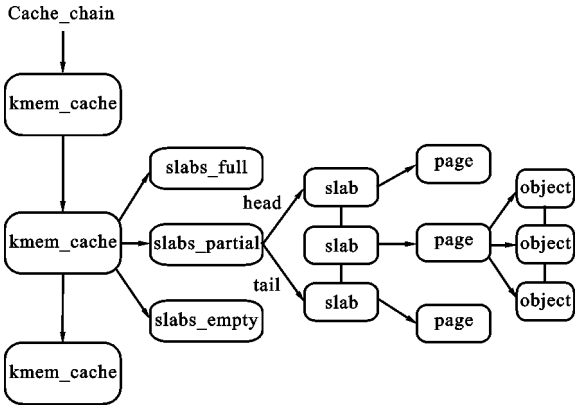

图 1　slab 分配器的主要结构

2　Linux 内存池

内存池是动态分配内存的设备，只能被特定的内核成分（即池的“拥有者”）使用。拥有者通常不直接使用内存池，当普通内存分配失败时，内核才调用特定的内存池函数来提取内存池，以得到所需的额外内存。因此内存池只是内核内存的一个储备，用在特定的时刻。这样做的一个显著优点是尽量避免内存碎片，使得内存分配效率得到提升[5]。

一个内存池通常叠加在 slab 分配器之上——也就是说，它被用来保存 slab 对象的储备。但是一般而言，内存池能被用来分配任何一种类型的动态内存，从整个页框到使用 kmalloc 函数分配的小内存区。因此，我们可以将一般内存池处理的内存单元看成“内存元素”。

引入了内存池之后，Linux 内核的内存管理采取两级分配机制，即初始化时将内存池按照内存的大小分成多个级别（每个级别均是 8 字节的整数倍数，一般是 8，16，24，…，128 字节），每个级别都预先分配了 20 块内存[6]。如果用户申请的内存单元大于预定义的级别（128 字节），则直接调用 malloc 从堆中分配内存。而如果申请的内存小于 128 字节，则从最相近的内存大小中申请，如果该组的内存存储量小于一定的值，就会根据算法，再次从堆中申请一部分内存加入内存池，保证内存池中有一定量的内存可以使用[7]。

3　Linux 内核中内存池的实现

Linux 内核中内存池的数据结构定义在＜linux/mempool.h＞中[8]。

3.1　内存池的数据结构

Linux 内核中内存池 mempool_s 的定义为

```
typedef struct mempool_s
{    spinlock_t lock;
     int min_nr;
     void  * *elements;
     void  *pool_data;
     mempool_alloc_t  *alloc;
     mempool_free_t  *free;
     wait_queue_head_t wait;
} mempool_t;
```

结构体中相应的字段描述如下：

lock：用来保护对象字段的自旋锁。

min_nr：内存池中元素的最小个数，elements 数组中的成员数量。

curr_nr：当前内存池中元素的个数，即当前 elements 数组中空闲的成员数量。

elements：用来存放内存成员的二维数组，其长度为 min_nr，宽度是上述各个内存对象的长度，因为对于不同的对象类型，会建立相应的内存池对象，所以每个内存池对象实例的宽度都是跟其内存对象有关的。

pool_data：内存池与内核缓冲区的结合使用，这个指针通常是指向这种内存对象对应的缓存区的指针。由于 Linux 采用 slab 技术预先为每一种内存对象分配了缓存区，每当我们申请某个类型的内存对象时，实际是从这种缓存区获取内存。

alloc：用户在创建一个内存池对象时提供的内存分配函数，这个函数可以用户编写，也可以采用内存池提供的分配函数。

free：与 alloc 功能相反的一个函数，内存释放函数。

wait：当内存池为空时使用的等待队列。

3.2　内存池的创建

内存池函数的实现在 mm/mempool.c 文件中定义。内存池初始化用到了 mempool_create 函数，它负责为一类内存对象构造一个内存池，传递的参数包括内存池分配的最小内存成员数量、用户自定义内存分配函数、用户自定义内存析构函数。该对象的缓存区指针，指向根据用户自定义内存分配函数所提供的可选私有数据。

mempool_create 函数的具体实现如下：

```
mempool_t  *mempool_create(int min_nr,
mempool_alloc_t  *alloc_fn,  mempool_free_t
```

```
*free_fn, void *pool_data)
    {
        return
    mempool_create_node(min_nr, alloc_fn,
    free_fn, pool_data, -1);
    }
```

函数的形参 min_nr 表示内存池中最少可以分配的对象数, alloc_fn 是由用户定义的对象分配函数名, free_fn 是由用户定义的释放对象函数名, pool_data 是用户定义对象分配函数可以访问的可选私有数据。这个函数生成并分配一个一定大小并事先已经分配的内存池。在 mempool_create 函数中调用了 mempool_create_node 函数。mempool_create_node 函数的实现在此不再赘述。

3.3 内存池的使用

如果需要使用已经创建的内存池, 则需要调用 mempool_alloc 函数从内存池中申请内存以及调用 mempool_free 函数将用完的内存归还给内存池。

3.3.1 内存池调用函数 mempool_alloc

内存池调用函数 mempool_alloc 的函数声明如下:

```
void *mempool_alloc(mempool_t *pool, int gfp_mask);
```

该函数实现时首先进行一些必要的定义和初始化, 并将用户传递进来的 gfp 进行配置。然后将通过形参传递进来的 gfp 掩码标志去掉_GFP_WAIT 和_GFP_IO 两个标志, 这样做的作用是试图调用用户自定义分配函数从缓存区申请一个内存对象, 而不是首先从内存池中分配, 如果申请不到, 再从内存池中分配。接着从内存池中获取一个内存对象, 若内存池不为空, 则返回给申请者一个内存对象。并且该函数实现中考虑了在 GFP_ATOMIC 事件发生时, 不能睡眠。

3.3.2 内存池的释放函数 mempool_free

内存池调用结束, 应调用 mempool_free 函数释放内存池, 也就是将内存对象重新放置到内存池中。mempool_free 函数声明如下:

```
void mempool_free(void *element, mempool_t *pool);
```

mempool_pool 函数第一个形参 element 是指申请的存放内存池成员的数组, 第二个形参 pool 指向由 mempool_create()函数分配的内存池。在 mempool_free 函数实现时, 如果当前内存池已满, 则直接调用用户内存释放函数将内存还给系统; 如果内存池还有剩余空间, 则将内存对象放入池中并唤醒等待队列。

4 内存池在网络文件系统中的应用

NFS(Network File System)是一种分布式文件系统, 允许网络中安装不同操作系统的计算机间共享文件和外设。内存池在网络文件系统中有着广泛的应用。

在 linux/fs/nfs/write.c 中, 首先定义一个 mempool_t 类型的指针变量 nfs_wdata_mempool:

```
static mempool_t *nfs_wdata_mempool;
```

然后就可以根据需要申请相应的数据空间, 其实现如下:

```
struct nfs_write_data *nfs_writedata_alloc
(unsigned int pagecount)
    {
        struct nfs_write_data *p = mempool_al-
loc(nfs_wdata_mempool, GFP_NOFS);
        if (p){
        memset(p, 0, sizeof( *p));
        INIT_LIST_HEAD( &p->pages);
        p->npages = pagecount;
        if (pagecount <= ARRAY_SIZE(p->
page_array))
        p->pagevec = p->page_array;
        else
    { p->pagevec = kcalloc(pagecount, sizeof
(struct page *), GFP_NOFS);
        if (!p->pagevec)
        { mempool_free(p, nfs_wdata_mempool);
        p = NULL;
        }
        }
    }
        free(p);
    }
```

当内存池调用结束, 再将内存对象重新放置到内存池中, 其实现如下:

```
void nfs_writedata_free(struct nfs_write_data
*p)
    {
        if(p && (p->pagevec != &p->page
_array[0]))
        kfree(p->pagevec);
```

```
    mempool_free(p, nfs_wdata_mempool);
}
```

5　结束语

分析了 Linux2.6 内核中内存池的数据结构及实现, 并对其在网络文件系统中的应用进行了研究。研究表明, 使用内存池可以减少内存碎片, 提高分配速度, 便于内存管理, 防止内存泄露。将内存池的分配方法应用于用户程序, 可提高效率。但是当分配的内存大小增大的时候, 使用内存池分配内存的速度有可能会有所降低。

参　考　文　献

[1] 汤小丹, 汤子瀛, 哲凤屏, 等. 计算机操作系统[M]. 西安: 西安电子科技大学出版社, 2007.
[2] 刘若慧, 毛莺池, 祁翔. Linux 操作系统[M]. 北京: 人民邮电出版社, 2008.
[3] 陈莉君, 康华. Linux 操作系统原理与应用[M]. 北京: 清华大学出版社, 2006.
[4] 吴国伟, 李张, 任广臣. Linux 内核分析及高级编程[M]. 北京: 电子工业出版社, 2008.
[5] 李云华. Linux 内核源代码导读[M]. 北京: 电子工业出版社, 2009.
[6] 肖竟华, 陈岚. Linux 内存管理实现的分析与研究[J]. 计算机技术与发展. 2007, 17(2): 187-189.
[7] 陈莉君, 康华, 张波译. Linux 内核设计与实现[M]. 北京: 机械工业出版社, 2006.
[8] 任桥伟. Linux 内核修炼之道[M]. 北京: 人民邮电出版社, 2010.

Implementation and application of the memory pool in Linux kernel

WANG Xiao-yin, CHEN Li-jun

(School of Computer Science and Technology, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

Abstract: After that the memory management pattern of Linux is analyzed, the definition of the memory pool in the kernel (version 2.6) of Linux is unscrambled, and the methods to create or transfer the memory pool are nailed down, besides, an example is given to show how the memory pool works in network file system. Experiments show that, when the memory pool is introduced for memory management into memory implementation and user program, the number of memory pieces will be reduced, the speed of distribution will be improved, and the leaks of memory will be avoided.

Key words: Linux kernel; memory pool; memory management

[责任编辑: 祝　剑]

(上接第 39 页)

[7] Dr. Rao Mikkilineni, Vijay Sarathy. Infrastructures for Collaborative Enterprises[C]//18th IEEE International Workshops on Enabling Technologies, 2009: 57-62.
[8] 朱珠. 基于 Hadoop 的海量数据处理模型和应用[D]. 北京: 北京邮电大学, 2008.

A method for frequent set mining based on MapReduce

RONG Xiang, LI Ling-juan

(School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: To improve the classic Apriori algorithm of the associate rules mining, a method for frequent set mining based on MapReduce is proposed. The new method can be implemented based on a platform of Hadoop, and thus, an experimental research can be given to compare the performance of the new method with that of the Apriori. The result shows that, the new method can take advantages of cloud computing to get good efficiency when mining mass data.

Key words: cloud computing; Hadoop; Apriori; MapReduce

[责任编辑: 孙书娜]