基于嵌入式系统的 e_slab 研究与实现

李 勇,任 宏,王志林
(中国人民解放军 91245 部队,辽宁 葫芦岛 125000)

摘 要:为降低嵌入式系统的内存管理开销,提升内存分配效率,详细分析了 slab 分配器机制并指出其不足,给出相应的改进措施,提出了基于 e_slab 算法的内存分配器。实验表明,e_slab 算法不仅简化了内存管理结构,而且提高了内存分配效率。

关键词:slab;e_slab;嵌入式系统;cache
中图分类号:TP316 **文献标识码**:A **文章编号**:1674-7720(2012)12-0071-03

Research and performance of e_slab based on embedded system

Li Yong, Ren Hong, Wang Zhilin
(The Troop 91245, PLA Army, Huludao 125000, China)

Abstract: To reduce the overhead of memory management in embedded system and improve the performance of memory allocation, slab algorithm is analyzed thoroughly and the shortcomings is pointed out, the appropriate improvements of slab algorithm and the memory allocation based on e_slab algorithm are proposed. The implementation shows that e_slab algorithm can simplify the structure of memory management and improve the efficiency of memory allocation.

Key words: slab; e_slab; embedded system; cache

随着硬件技术的发展和内存容量的扩大,操作系统中内存管理技术日趋完善。但是在嵌入式领域中,硬件性能和内存容量远远落后于 PC 机,其内存管理受到多种因素制约,若直接采用操作系统中的内存管理技术,不仅难以达到预期效果,而且会影响嵌入式系统的性能。

在嵌入式系统内存管理设计过程中,发现操作系统中的 slab 分配器虽然在 PC 机上有良好的性能,但是在嵌入式系统中不但不能发挥其优势,还降低了系统的整体性能。本文通过分析,指出了 slab 分配器的不足,并给出相应的解决方案。实验结果表明,slab 分配器经过改进可适用于嵌入式系统。

1 slab 分配器分析

操作系统内核运行时会频繁地为某些对象分配内存空间,而这些对象往往只需要几十或几百 KB 的空间,如果直接采用页面管理器进行内存分配,将产生很多内存碎片,造成严重的内存浪费。slab 分配器支持细粒度的内存分配,较好地解决了此问题。由于性能优越,slab 被 Linux、FreeBSD 等操作系统采用,是目前应用最广的内核内存管理器之一[1]。

1.1 slab 分配器设计思想

基于页面分配器[2],将一页或几页的内存组织起来,划分成一定数量的小块内存,这种连续的页面称之为 slab。它为内核中使用频繁的对象建立专门的缓冲区(cache),每种类型的对象都有自己专用的 cache[2]。一个 cache 管理着多个 slab,每个 slab 又管理着多个对象。slab 的大小与所管理对象的大小有关。根据 slab 管理对象的分配情况,可将每个 cache 中的 slab 分为 3 类[3-4]:(1)slab 管理的对象已经完全分配,没有空闲的对象;(2)slab 管理的对象部分分配,还有部分空闲对象;(3)slab 中的对象都未分配,都是空闲对象。

不同的 slab 分别放入不同的队列中,即每个 cache 管理 3 个 slab 队列,cache 与 cache 之间的关系如图 1 虚框①内所示,cache 与 slab 的关系如图 1 虚框②内所示。

当 slab 分配器接收到内存申请时,根据所申请内存的大小找到合适的 cache,从 cache 管理的第二类 slab 中分配对象,若失败则从第三类 slab 中分配对象,若还不成功则说明 cache 中没有空闲对象,须为 cache 创建一个新的 slab,从新的 slab 中分配空闲对象。

对象释放过程中,不仅要清空对象占用的空间,而

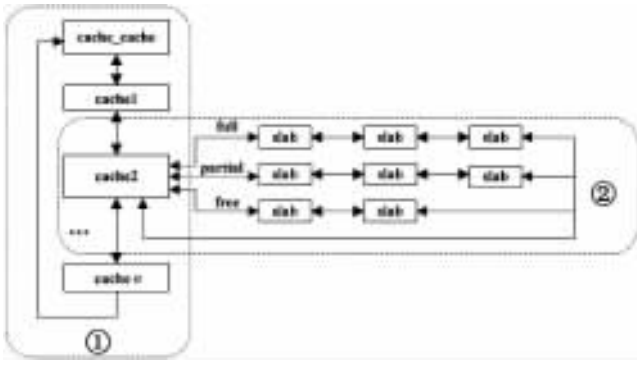

图 1 cache 与 slab 组织结构

且还要调整对象所属 slab 的状态，判断是否改变此 slab 在 cache 中的位置。

slab 分配器采用着色机制将不同 slab 中的对象放入不同的偏移处，利用硬件高速缓存的映射机制，将页的不同偏移映射到硬件缓存的不同地址。而每个 slab 的开始部分访问频率最高，只要 slab 中起始对象的偏移不同则映射到硬件高速缓存的位置就不同，从而降低了频繁换入换出的性能损失[4-5]。

1.2 slab 分配器在嵌入式系统中的缺陷

slab 分配器虽然能解决系统对小块内存的频繁需求，但是管理结构复杂，内存分配策略开销较大。在内存受限的嵌入式系统中，slab 的缺陷大大影响了系统的整体性能。总之，slab 分配器存在以下三方面的缺陷：

(1)slab 管理结构和存储开销较大

每个 slab 由 slab 描述结构、管理空闲对象的整型数组和对象三部分组成，整型数组把 slab 中空闲对象组成一个顺序队列，数组大小与对象数有关，每个对象对应一个整数，如图 2 所示。当对象较小时，整型数组将造成较大的内存开销。

图 2 slab 内部结构

(2)cache 结构复杂而且数量较多

系统中存在着专用对象和通用对象。专用对象专门存储特定用途的数据结构，例如 CPU、文件系统等，其数量与系统密切相关；通用对象用来存储一般的数据结构，大小在几十 KB 到几千 KB 之间(一般为 2 的整次幂字节)，有十多种。不管是专用对象还是通用对象，slab 分配器都为其建立了一个 cache 结构，众多 cache 组织和管理的较大开销是嵌入式系统难以承受的。

(3)复杂的队列管理

如图 1 所示，slab 分配器中存在较多的队列，每个 cache 管理着 3 个 slab 队列，每个 slab 队列与 cache 组成循环队列。所有的 cache 组成双向循环队列。面对众多的队列，如何有效地管理是很困难的。

1.3 slab 在嵌入式系统中的改进

针对上节中 slab 分配器的三点缺陷，给出相应的改进方案。

(1)改进 slab 结构

针对 slab 中对象管理数组开销过大的问题，可以将多个不同的 slab 合并成一个 slab，从而减少 slab 的数量，即一个 slab 管理对象的大小可在一个小范围内浮动。由于 slab 中对象大小不同，无法确定 slab 中对象的大小、数量和位置，所以必须重新设置 slab 结构。

(2)限制 slab 分配器管理的内存粒度范围

由于内核内存管理器主要负责细粒度的内存管理，所以限制所管理对象的大小。对于大块内存的申请，直接由页面分配器处理。

(3)精简队列管理

简化 cache 中繁杂的队列，将 cache 中的前两个 slab 队列合并成一个队列。

本文将经过上述三方面改进的分配器称之为 e_slab 分配器。

2 e_slab 分配器设计

2.1 基本管理结构

e_slab 分配器有 3 个重要的基本结构，下面分别对其作相关介绍。

(1)object_t 结构

```
typedef struct object {
        unsigned long size;
        unsigned long offset;
} object_t;
```

object_t 是描述对象的基本结构，每个对象对应一个 object_t 结构，它描述了对象的大小和下一个空闲对象的地址。

(2)e_slab_t 结构

```
typedef struct e_slab _s {
        struct list_head    list;
        void                *s_mem;
        unsigned int        units;
        unsigned int        free;
} e_slab _t;
```

e_slab _t 是管理对象的基本结构，它不仅描述了本结构的页块起始地址，而且存储了空闲对象的数量和地址等信息。

object_t、e_slab _t 和对象结构如图 3 虚框②内所示。

(3)cache 结构

```
typedef struct cache_s {
        struct list_head    next;
        struct list_head    slab_list;
        unsigned int        objsize;
```

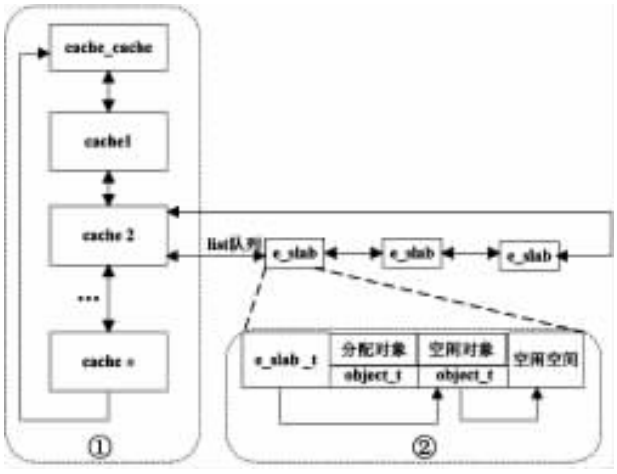

图 3 cache 与 e_slab 组织结构

```
    unsigned int        gfporder;
    unsigned int        num;
    ...
} cache_t;
```

cache 的描述结构为 cache_t，它主要描述了所管 e_slab 的基本信息。由于 cache_t 结构大小相同，可把 cache_t 看做一个专用对象，所有的 cache 组织在一起。

cache 管理的所有 e_slab 被加入到 list 队列。把管理所有 cache 的结构称之为 cache_cache。cache_cache 与 cache 之间有两种关系：一种是双向队列关系，如图 3 虚框①内所示，cache_cache 利用双向链表将系统中所有的 cache(包括专用 cache 和通用 cache)组成循环队列；一种是 cache 与对象之间的关系。

2.2 e_slab 分配器初始化

e_slab 分配器初始化主要完成 cache、e_slab_t 等结构的创建，为对象的分配做好准备。

2.2.1 cache 的创建

cache_cache 是系统中所有 cache 的管理者，它的创建优先于所有的 cache。系统会为每种对象创建一个 cache，创建流程如下：

(1)申请一个 cache 空间。从 cache_cache 中分配一个专用对象，即 cache。设置 cache 中的各个域，包括管理的对象大小的上限和 e_slab 大小，其中 e_slab 大小与对象大小有关。

(2)将此 cache 加入管理队列。将 cache 加入 cache_cache 组成的双向队列中，双向队列采用通用链表链接所有的 cache。

(3)将 cache 加入分配队列。用一个全局的 cache 指针指向生成的 cache。

2.2.2 e_slab 的创建

在 cache 的创建过程中，需为每个 cache 创建一个 e_slab。e_slab 中的对象全部空闲，可供分配，其流程如下：

(1)借助页面分配器申请连续物理页面。根据 e_slab 申请的大小和是否有 DMA 请求，在相应的内存区申请连续页块。

(2)设置页面属性。主要设置该页面的 e_slab 标志，并将该页块与 cache 和 e_slab 关联。

(3)设置 e_slab 描述结构，初始化对象结构。

2.3 对象的分配与释放

对象的分配与释放是内存管理模块提供的两个基本接口。

2.3.1 对象的分配

当系统需要小内存块或者专用对象时，系统会调用对象分配操作，完成对对象的分配，具体流程如图 4 所示。

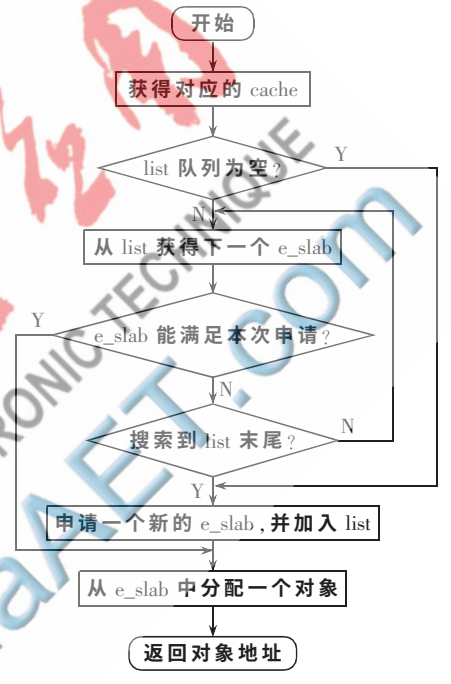

图 4 对象分配流程

(1)找到对应的 cache。根据申请对象的大小定位相应的 cache。

(2)确定对应的 e_slab。检查 cache 中的 e_slab，找到满足本次请求的 e_slab，如果所有的 e_slab 均不能满足，则创建一个新的 e_slab 并添加到 cache 管理的队列中。

(3)从 e_slab 中分配一个空闲对象。从 e_slab 为系统分配一个空闲对象和 object_t 结构，将对象返还给系统，调整 e_slab 中对象，管理数组结构。

2.3.2 对象的释放

系统使用完对象后，应及时释放对象，否则内存会越用越少。对象释放流程如图 5 所示。

(1)确定对象对应的 cache 与 e_slab。根据对象的地址可以获得所在页面的描述符结构，从而获得对应的 cache 和 e_slab。

(2)释放对象。获得对象在 e_slab 中的偏移，采用头插法将对象加入空闲对象队列，并使 e_slab 中空闲内存增加释放值。

(3)e_slab 的调整。检查 e_slab 中空闲内存大小，若等于 e_slab 中所有对象都释放，则清除页面的 e_slab 标志，并把 e_slab 占用页块归还给物理内存管理器。

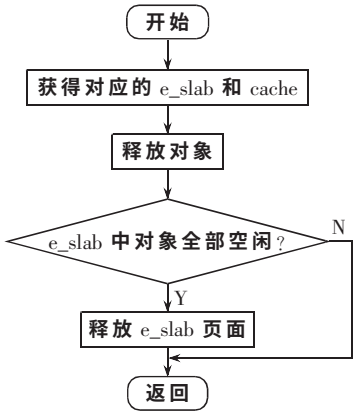

图 5 对象释放流程

2.4 e_slab 分配器的回收

在系统退出、内存回收等不再需要 e_slab 分配器时，需进行 e_slab 分配器的回收，主要完成 e_slab 的释放和 cache 的释放。

2.4.1 e_slab 的释放

在对象释放过程中，若发现某个 e_slab 已经全部空闲，没有分配的对象，则将其释放，流程如下：

(1)将 e_slab 从 cache 结构中删除。e_slab 从 cache 的 list 队列中摘掉。

(2)清除页面标志。将 e_slab 所在物理页面的 e_slab 标志清除，并清除页面与 e_slab 和 cache 的关联，使页面回到初始状态。

2.4.2 cache 的释放

当系统不再使用某种对象时，系统要销毁管理对象的 cache。cache 销毁流程如下：

(1)将 cache 从管理队列摘掉。将 cache 从 cache_cache 组成的双向队列中删除。

(2)确定 cache 中没有 e_slab。在 cache 销毁前，必须确定所管理的对象都已释放，检查 cache 的 list 队列，为空则 cache 中没有 e_slab，否则进行 e_slab 释放。

(3)释放 cache 结构占用的内存。由于 cache 是 cache_cache 管理的对象，cache 结构的释放过程就是对象的释放过程。

3 性能测试

在嵌入式系统内存管理设计过程中，分别采用页面分配器与 slab 分配器相结合的方案和页面分配器与 e_slab 分配器相结合的方案，比较两种方案中 slab 和 e_slab 管理结构的内存占用量和内存分配释放中的性能。

在管理结构内存占用方面，e_slab 比 slab 节省了 43%的空间；在对象的内存申请过程中，e_slab 的速度比 slab 快 8%；内存释放过程中，e_slab 比 slab 快 5%。可见，不管在时间上还是空间上，e_slab 性能都比 slab 优越。

参考文献

[1] GORMAN M.Understanding the Linux virtual memory manager[M].北京：北京航空航天大学出版社，2006.

[2] 陈燕晖.页面分配器的研究与实现[D].长沙：国防科技大学，2006.

[3] 李毅.Slab 内存分配策略与移植[D].成都：电子科技大学，2007.

[4] 李勇.虚拟机监控器内存管理机制研究与实现[D].郑州：信息工程大学，2010.

[5] 史成伟.多核系统中的内存管理系统优化研究[D].成都：电子科技大学，2009.

(收稿日期：2012-02-20)

作者简介：

李勇，男，1983 年生，硕士研究生，主要研究方向：操作系统，虚拟化，系统安全。