一种嵌入式系统内存管理的延迟合并伙伴机制

郭振宇，桑 楠，杨 霞

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】为提高嵌入式系统动态存储管理机制的运行效率，确定时间开销，该文在分析经典伙伴系统的基础上，提出了延迟合并的方法，并辅以碎片整理及位图机制。通过仿真试验效果分析，该机制具有更好的运行效率。

关 键 词 位图机制； 伙伴系统； 碎片整理； 延迟合并

中图分类号 TP316.2 文献标识码 A

A Recombination-Delaying Buddy Mechanism for Embedded System Memory Management

GUO Zhen-yu，SANG Nan，YANG Xia

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 610054)

Abstract In order to improve the runtime efficiency of dynamic storage management for embedded system and to bound the time spending, a method of recombination-delay is presented based on the classic buddy system, fragmentation collection, and bitmap mechanism. Simulation experiments show that this new mechanism has better runtime efficiency.

Key words bitmap mechanism; buddy system; fragmentation collection; recombination-delay

随着嵌入式产品开发复杂程度的增加，如何在嵌入式系统中更加有效地管理动态内存显得越发重要。如果内存短缺或者管理不当，将导致整个系统反应迟缓，甚至崩溃。现代操作系统已经运用位图、链表等多种手段来管理内存。但在嵌入式环境下，还要求内存管理机制具有运行效率高、时间开销可确定的特点。为此，本文在伙伴系统的基础上，提出了延迟合并的方法。

1 伙伴算法描述

由文献[1]提出的伙伴系统是一个快速的动态内存管理经典算法(下文称其为经典伙伴系统)。在该算法中有多个空闲队列，块长为2^k个页面的空闲块都链在同一个队列中。当要分配一个长为d的内存块时，求i使得$2^{i-1}<d\leq 2^i$，然后从长为2^i个页面的空闲队列中分配一块内存。如果该空闲队列耗尽，则从长为2^{i+1}个页面的空闲队列中分配一个块，将其分为长度相等的两半(互为伙伴)，一半用于分配，另一半链入长为2^i的伙伴空闲队列中。如果长为2^{i+1}个页面的空闲队列也空，就继续请求更大的内存块(2^{i+2}, 2^{i+3},…)。

当长度为2^k个页面的内存块被释放时，首先检查其伙伴块是否空闲，如果空闲则与伙伴块合并为长2^{k+1}个页面的空闲块并链入对应队列中；若忙则直接链入长为2^k个页面的空闲队列中。这个过程是递归的，即合并得到的2^{k+1}的空闲块也要检查它的伙伴块状态并做相同的处理，直到得到最大的空闲块2^M。

经典伙伴系统是一个非常强调时效的算法。最坏情况下内存分配和回收的时间开销都是$O(\log_2 M)$，M是堆的大小。实时嵌入式系统的堆大小M可以假定为固定的，这样内存的分配和回收可在一定时限内完成。经典伙伴系统内存管理的主要时间消耗在分配时没有合适的内存块时，把大的内存块分裂成较小的内存块的过程，以及在回收时，把小的空闲块合并成较大空闲块的过程。

虽然经典伙伴系统有上述优点，但它在嵌入式系统中却没有得到广泛的应用，原因有两点：(1) 经典伙伴系统使内存碎片化[1]，这意味着即使堆中有足够内存，但其地址不连续从而不能满足某些内存请求。(2) 若创建对象的生存周期很短，经典伙伴系

收稿日期：2005－06－13
基金项目：“十五”预研项目(4131504040b)
作者简介：郭振宇(1981－)，男，硕士，主要从事实时操作系统防危技术方面的研究.

统的运行效率较低。在这些情况下，系统因为不断地将内存块分裂合并而产生过大的负载。

针对上述问题，本文提出了一种采用延迟合并、辅以碎片整理和位图机制的伙伴系统。

2 重要机制

2.1 延迟合并

所谓延迟合并是指当一个内存块被释放时，并不立即合并。这个思想首先由文献[2]提出，但是它没有给出具体的实现。本文针对现在常用的面向对象语言所面临的问题，提出了具体的实现方法。

面向对象程序倾向于分配较小的块，通过较少的属性描述简单的对象。这种情况会造成经典伙伴系统不断分裂和合并内存块，引起内存振荡，同时面向对象程序也倾向于请求许多相同大小的块。如在Java中，同样的类型代表同样大小的内存请求(数组除外)，因此对于内存块的请求意味着将来还会有同样大小的请求。延迟合并策略使得内存释放后不必立刻进行合并，既有足够的空闲块来满足下一次的同种内存请求，又可减少内存振荡。为此，在延迟合并策略中，将2^k个页面的空闲队列分为伙伴忙队列和伙伴闲队列，如图1所示。其中伙伴忙空闲队列包含其伙伴未被释放的空闲块，而伙伴闲空闲队列则包含伙伴块已释放的空闲内存块。注意，该链表中只包含互为伙伴的两个空闲块中的一个，另一个可根据该块获得。

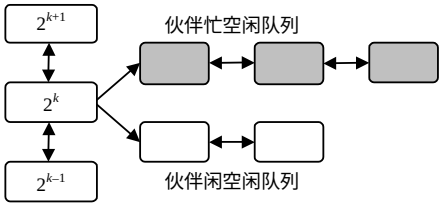

图1 延迟合并伙伴队列结构

显然，当请求长为2^k的空闲块时，首先应该分配伙伴忙空闲块链中的块，当其为空时才考虑分配伙伴闲空闲链表中的块，因为后者可以合并成为更大的块供程序使用。在内存回收时，根据其伙伴块状态将它链入相应的空闲链表中，而不需要立即进行合并，等到内存请求不能立即得到满足时才尝试合并。

2.2 外部碎片整理

实时嵌入式系统内存整理算法必须有两个特性：(1) 碎片整理时间必须在限定时间内。(2) 堆大小不能被扩展，当嵌入式系统交付使用时，堆的大小就已经固定。

针对上述的问题，本文提出一种在内存请求不能满足时使用的内存整理算法，其开销较小。该算法能整理请求大小的空闲块给程序使用，并不整理所有的系统碎片。算法的关键思想是将伙伴忙空闲队列中的块链入到伙伴闲队列，再不断合并得到请求大小的块。如图2所示，伙伴忙空闲链表中有三个块，它们的伙伴都已被分配且还没有回收，算法通过将第一个空闲块的伙伴与第二个空闲块相交换，结果是第二个空闲块被占用，移出空闲链表，而第一个空闲块的伙伴块也变为空闲，因此第一个空闲块可以被移入伙伴闲空闲链表，这样就可以通过合并取得较大的内存块，再通过递归调用这一算法就可以获得2^k大小的内存块。

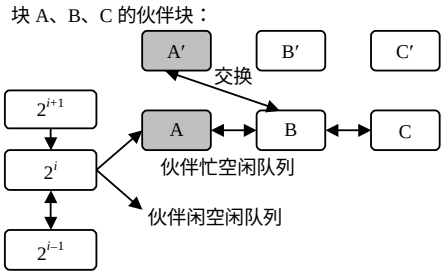

图2 外部碎片整理算法($0 \le i \le k-1$)

2.3 位图机制

经典伙伴系统的大量时间消耗在不断分裂和合并内存块的过程中。为了加快这一过程，本文提出了一个基于位图机制的管理方法。加入位图机制后，伙伴系统的manage_struct结构定义如下：

```
struct    manage_struct{
        struct manage_struct *prev, *next;
        struct                          mem_buddy
    membuddy[MAXBUDDY][2]; };
```

这里的membuddy[MAXBUDDY][2]二维数组用于管理不同页面长度的空闲链表，页面长度与数组的第一维下标成正比，数组的第二维下标表示该长度下的一个空闲链表(0标识伙伴忙空闲链表，1标识伙伴闲空闲链表)。如，membuddy[6][0]管理着一条长为2^6个页面的伙伴忙空闲链表。

mem_buddy的主要数据结构定义如下：

```
struct mem_buddy{
        struct page     *freearea;
        unsigned long *bitmap, count;  … };
```

其中：freearea是指向page结构的指针，page结构使不同伙伴块形成一个双向链表；bitmap是位图指针，伙伴系统通过位图来标识当前内存的使用状况。位图的每一位都描述了一对伙伴块的状态：位图中0

表示一对伙伴中要么两个都被分配，要么都空闲；1则表示这对伙伴中有一个已经被分配。

引入位图的目的在于加速查询伙伴块的状态。每个数组对应的位图表示当前内存分配状况。系统初始时，位图中全部位均为0。当一个伙伴块释放时，它所在的位图位就会产生一次异或操作。假定该位原来是0，当某一块释放时就会产生一次异或操作，该位变成1，表示该对伙伴块中至少有一个忙，相应将该空闲块链入伙伴忙空闲队列；如果接着这个块的伙伴块被释放了，该位再进行一次异或操作，又将1变成0，表示这对空闲块都闲，于是将先前链入在伙伴忙空闲链表中的块链入伙伴闲空闲链表。

3　算法描述

分配的核心是一个递归函数，通过它可获得要分配的物理块地址，其流程用类C形式化描述如下：

(1) RequestSize_temp = RequestSize; /*暂存请求大小*/

(2) I = GetLinklist(RequestSize);/*根据请求内存大小确定寻找的链表*/

(3) if(membuddy[*i*][0].freearea!=NULL) goto 11);

(4) elseif(membuddy[*i*][1].freearea!=NULL)goto 11);

(5) elseif(membuddy[*i*–1][1] != NULL){

(6) membuddy[*i*–1][1].freearea.first+Buddy (membuddy[*i*–1][1].freearea.first)→membuddy[i][0].freearea.first; goto 3); }　/*将第 *i*–1 伙伴闲链表的首个内存块及其伙伴合并为第 *i* 伙伴忙链表中的首个内存块*/

(7)else{RequestSize=RequestSize∗2;if(*i*<= MAXBUDDY)goto 2); } /*向更大的链表寻找空闲块*/

(8) if(Defragmentation(*i*) = TURE){

(9) membuddy[*i*–1][1].freearea.first+Buddy (membuddy[*i*–1][1].freearea.first)→membuddy[*i*][0].freearea.first; goto 3); }

(10) return FALSE; /*分配失败，返回 FALSE*/

(11) while(RequestSize_temp < 2^{i-1}) {

(12) membuddy[*i*][0].freearea.first→membuddy[*i*–1][1].freearea.first+Buddy(membuddy[*i*–1][1].freearea.first);

(13) ModifyBitmap();　*i* = *i*–1; }　/*修改相应位图信息*/

(14) else{ModifyBitmap();return membuddy[*i*][0] .freearea.first;} /*分配成功，返回分配的空闲块地址*/

外部碎片整理算法Defragmentation()描述如下：

(1) for(*j* = *i*–2; *j* >= 0; *j*--) {temp = *j*;

(2) if(membuddy[*j*][1] != NULL) {

(3) membuddy[*j*][1]. freearea.first+Buddy (membuddy[*j*][1].freearea.first)→membuddy[*j*+1][0].freearea.first;

(4) if(*j* == *i*–2) return TRUE; else{*j* = *j* +1; goto 2}; } }

(5) elseif(Listlength(membuddy[*j*][0].freearea)> 1) { /*伙伴忙空闲队列长度大于 1*/

(6) Buddy (membuddy[*j*] [0] .freearea.first) ←→ membuddy[*j*][0].freearea.first.next;　/*内存块交换*/

(7) membuddy[*j*] [0] .freearea.first+Buddy (membuddy[*j*][0].freearea.first)→membuddy[*j*][1].freearea.first; goto 2); }

(8) ModifyBitmap();　*j* = temp; }

(9) return FALSE;

延迟合并伙伴系统在回收时并不立即将内存块合并，只需根据伙伴块状态将其链入相应队列即可。

4　试验结果

基于上面对伙伴系统的描述，本文通过仿真试验进行了验证，主要考虑了内存分配的时间开销和成功概率这两个主要性能指标。仿真试验平台为：CPU：Intel (R) Celeron (TM) 400 MHz；内存：128 MB BSDRAM。

4.1　最差情况下的性能比较

最坏情况主要根据如下过程：(1) 分配*N*个大小为2^k的对象，占据大部分的堆空间(还剩下一个大于2^k的残留块)。(2) 释放所有序号为偶数的对象。(3) 垃圾收集周期开始，使得空闲链表包含*N*/2个大小为2^k的块，且有一个比2^k更大的残留块在末尾。(4) 一个更大的(大于2^k)对象分配请求发生。

上述的分配过程使得无结构链表要不断搜索，直到搜到链表的结尾才能找到大小合适的分配块。而伙伴系统则可以直接通过将大小为2^k的空闲块合并得到合适的块，因此伙伴系统运行更快。在本文试验中，当分配对象个数为3 000时，伙伴系统的分配速度是无结构链表的72倍。

4.2　一般情况下性能比较

通过Java SPEC测试，本文对无结构链表(采用JDK1.1.8)、经典伙伴系统和本文改进后的伙伴系统

进行了对比，结果如表1所示，延迟合并伙伴系统的性能比无结构链表分配策略大约高4%，也比经典算法优越。

表1 伙伴系统与无结构链表在平均情况下的性能比

测试方法	无结构链表	延迟合并伙伴系统	经典的伙伴系统
jess	1.00	1.04	1.02
raytrace	1.00	1.03	0.92
db	1.00	1.02	1.01
javac	1.00	1.02	1.00
jack	1.00	1.05	1.03

4.3 延迟合并伙伴系统的定性分析

本文仿真运行了经典伙伴系统和延迟回收伙伴系统，通过跟踪每次内存分配的过程，定性地分析了请求的满足状况。数据显示延迟合并伙伴系统有约90%的机会从伙伴忙或者伙伴闲空闲链表中直接得到满足。其成功之处在于每次分配的都是相对较小的内存块；而经典伙伴系统只有50%的机会立即找到合适的空闲块。

5 结 论

本文在分析经典伙伴系统的基础上，针对面向对象语言特点提出了延迟合并的伙伴系统。通过延迟合并机制使得约90%的内存分配请求能够立即得到满足，并降低了合并造成的系统开销；通过外部碎片整理能够以较少的负担实现内存整理的功能，从而降低内存分配失败的机率；通过位图机制加速了伙伴块的查询过程，从而加速了伙伴系统的分配和回收流程。仿真试验表明延迟合并伙伴系统与经典伙伴系统相比具有更高的执行效率，有更多的机会能够马上找到合适大小的空闲块以满足请求。

参 考 文 献

[1] KNUTH D E. 计算机程序设计艺术, 第1卷: 基本算法[M]. 第3版. 苏运霖, 译. 北京: 国防工业出版社, 2002.
[2] ARIE K. Tailored-list and recombination-delaying buddy systems[J]. ACM Transactions on Programming Languages and Systems, 1984, 6(1): 118-125.
[3] 曹全欣. 动态存储管理机制的改进及实现[D]. 南京: 南京航空航天大学, 2003.
[4] LO C-T D, SRISA-AN W, CHANG J M. Performance analyses on the generalized buddy system[J]. Computers and Digital Techniques, IEEE Proceedings, 2001, 148(45): 167-175.
[5] 郭福顺, 王世铀, 臧天仪. 一种动态存储管理机制[J]. 计算机研究与发展, 1999, 36(1): 62-66.

编 辑 漆 蓉

(上接第540页)

6 结 论

本文提出的两方安全议价协议具有较高的执行效率，其通信复杂度和计算复杂度均为常数阶，并能够提供私密性和公平性的保障，因此在电子拍卖、公平交换等电子商务中具有较好的实用价值。目前该协议适用于两方的价格协商，在下一步的工作中将进一步研究将其拓展到多方的方法和途径。

参 考 文 献

[1] YAO A C. Protocols for secure computations[C]// Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science. Chicago: IEEE Computer Society, 1982: 160-164.
[2] GOLDREICH O. Secure multi-Party computation[EB/OL]. http://www.wisdom.weizmann.ac.il/~oded/pp.html, 2006-08-06.
[3] GOLDREICH O, MICALI S, WIGDERSON A. How to play any mental game[C]//Proceedings of the 19th Annual ACM Symposium on the Theory of Computing. New York: ACM Press, 1987: 218-229.
[4] GOLDREICH O, MICALI S, WIGDERSON A. Proofs that yield nothing about their validity -or- all languages in NP have zero-knowledge proof systems[J]. Journal of the ACM, 1991, 8(1): 691-729.
[5] FISCHLIN M, EFFECTIVE C A. Pay-per-multiplication comparison method for millionaires[C]//RSA Security 2001 Cryptographer’s Track at RSA Conference. LNCS2020, London: Springer-Verlag, 2001: 457-471.
[6] PAILLIER P. Public-key cryptosystems based on Composite degree residuosity classes[C]//Proceedings of Eurocrypt'99, Prague, Czech Republic, LNCS1592. Berlin: Springer-Verlag, 1999: 223-238.
[7] CATALANO D, GENNARO R, GRAHAMN H. The bit security of paillier encryption scheme and its applications[C]//In Advances in Cryptology-Eurocrypt '01, Aarhus, Denmark, LNCS2045. Berlin: Springer-Verlag, 2001: 229-243.
[8] CRAMER R, SHOUP V. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption[C]//In Advances in Cryptology-Eurocrypt '02, Amsterdam Netherlands, LNCS 2332. Berlin: Springer-Verlag, 2002: 45-94.

编 辑 孙晓丹