28卷 第11期
2011年11月

微电子学与计算机
MICROELECTRONICS & COMPUTER

Vol. 28 No. 11
November 2011

嵌入式操作系统 μC/OS－Ⅱ的一种内存管理算法

李平勇[1]，游 磊[1,2]

（1 成都大学 信息科学与技术学院，四川 成都 610106；
2 成都理工大学 核技术与自动化工程学院，四川 成都 610059）

摘 要：针对 μC/OS－Ⅱ内存管理机制的不足，提出了一种新的内存管理算法. 较小的内存分成固定大小的内存块，用位图索引组织；较大的内存用链表组织. 实验表明，该方法能较好地提高内存分配速度和利用率，特别是对于内存块大小变化很大的系统.

关键词：μC/OS－Ⅱ；内存管理；内存分区；内存块；动态内存分配

中图分类号：TN911 **文献标识码**：A **文章编号**：1000－7180(2011)11－0098－04

A Memory Allocation Algorithm of Embedded Operating System μC/OS－Ⅱ

LI Ping-yong[1], YOU Lei[1,2]

(1 College of Information Science and Technology, Chengdu University, Chengdu 610106, China;
2 College of Applied Nuclear Technology and Automation Engineering,
Chengdu University of Technology, Chengdu 610059, China)

Abstract: In order to solve the fault of μC/OS-Ⅱ memory management, this paper proposes a new algorithm of memory management. The smaller memory is divided into fixed－size memory blocks and organized by bitmap index, while the larger memory is organized by list. Experimental results show that this method can improve the memory allocation speed and memory utilization, especially for large changes of memory block size.

Key words: μC/OS-Ⅱ; memory management; memory partition; memory block; dynamic memory allocation

1 引言

μC/OS-Ⅱ是基于优先级的抢占式实时多任务嵌入式操作系统，简洁实用，对系统硬件要求很低. μC/OS-Ⅱ能满足很多项目的需求，其稳定可靠的特性已得到美国航空管理局(Federal Aviation Administration)认证，成功应用于医疗科学、航天工程等重大科研项目中. 由于嵌入式系统资源有限，其中内存管理一直是嵌入式操作系统中一个非常重要的研究课题，尤其是动态内存分配和释放算法，至今没有一个很好的解决方案. 文中以嵌入式操作系统 μC/OS－Ⅱ为例，在原有内存分配算法的基础上，提出了一种新的内存分配方案，以满足嵌入式系统对内存快速、高效、可靠的要求[1].

2 μC/OS－Ⅱ内存管理机制分析及存在的问题

μC/OS－Ⅱ对内存进行两级管理[2]，即把一个连续的内存空间分为若干个分区，每个分区又分为若干个大小相等的内存块. 操作系统以分区为单位来管理动态内存，而任务以内存块为单位来获得和释放动态内存. 内存分区及内存块的使用情况则由内存控制块来记录. 不同的内存分区之间管理的内存块大小没有限制，可以相同也可以不同. 内存控制块与内存分区和内存块的关系如图1所示[3].

嵌入式操作系统 μC/OS－Ⅱ的内存管理方案

收稿日期：2011－01－15；**修回日期**：2011－02－10
基金项目：成都大学校科技基金项目(2009XJ11)

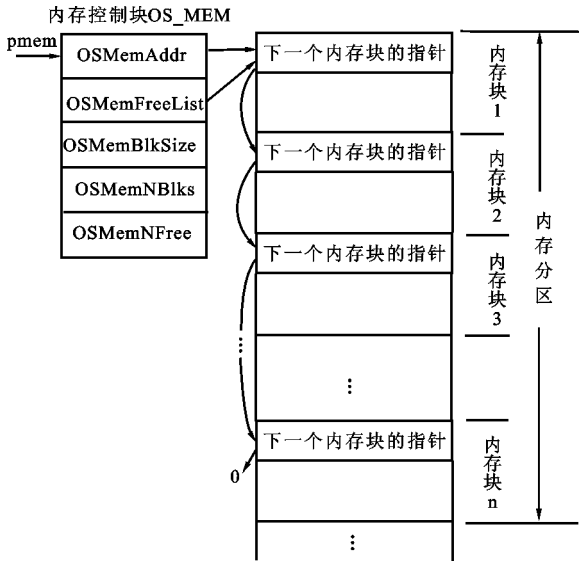

图1　内存控制块与内存分区和内存块的关系

虽然简单易行，分配速度快，适合在嵌入式系统中使用，但存在以下不足：

（1）创建内存分区后内存块大小固定，使用要求严格

μC/OS－Ⅱ创建内存分区时采用了定义二维全局数组的方法，将数组名作为内存分区的首地址. 数组大小是固定的，生成映像后不可能在使用中动态地改变. 生成的内存分区中内存块由于大小固定，要求开发人员事先知道该分区中内存块的大小，并且在使用时不能超过该内存块的长度，而且在释放内存块时，一定要确保该内存块释放到它原来所属的内存分区中，否则会引起灾难性后果.

（2）未提供内存分区的释放和合并功能

基于效率和简单性的考虑，μC/OS－Ⅱ的内存块虽然可以释放，但没有提供内存分区释放功能，造成这部分内存空间一旦被任务申请使用后就无法再次回收利用. 这对于一些稍微复杂一点应用环境来说是一种比较苛刻的条件，可能会导致产品对内存需求增大，从而引起成本的增加.

μC/OS－Ⅱ的内存分配方案使系统失去了灵活性，要求必须在设计阶段就预先知道所需要的内存并对之作出分配. 这样的分配方案必然导致很大的浪费，因为内存分配必须按照最坏情况进行最大的配置，而实际运行时很可能只使用其中的一小部分，而且在硬件平台不变的情况下不可能灵活地为系统添加功能，从而使得系统的升级变得困难.

3　μC/OS－Ⅱ内存管理新算法设计

为了提高分配内存块的效率并且减少碎片的产生，对不同大小的内存分配请求使用不同的策略[4]. 文献[5]表明，在大多数系统中，小块内存的分配请求远多于对大块内存的分配请求，因此对小块内存分配请求应该采用高效、快速的分配方法.

3.1　小块内存管理方案

当需要的内存较小时（≤1KB），使用一级位图索引[6]，把内存按2的幂次分为固定大小的块. 由于在很多32位的嵌入式系统中内存是8Byte对齐，因此最小的内存块为8Byte，然后依次递增，即8Byte，16Byte，…，1KB，一共8组.

定义一个存储内存控制块首地址的指针数组 void ＊pArray[7－8]，指针数组初始化为 NULL. 指针数组 pArray 第1个元素的值为指向第1个内存控制块的指针，这个内存控制块所控制内存分区的内存块大小为8Byte；指针数组 pArray 第2个元素的值为指向第2个内存控制块的指针，这个内存控制块所控制内存分区的内存块大小为16Byte；以此类推……，指针数组 pArray 第8个元素的值为指向第8个内存控制块的指针，这个内存控制块所控制内存分区的内存块大小为1KB. 如图2所示.

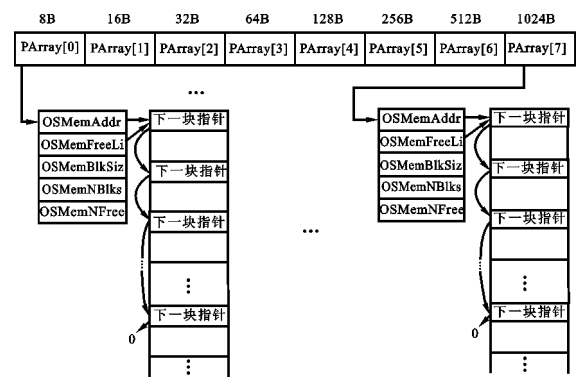

图2　小块内存管理中存储内存控制块首地址的指针数组

当应用程序向系统申请一个内存分区时，先从空内存控制块链表中摘取一个控制块，再根据内存分区中内存块的大小，把这个内存分区的内存控制块首地址放入到这个指针数组的相应位置. 申请内存块的大小一般不会恰好等于某个固定内存块的大小，应换算成对应的固定内存块大小.

具体换算过程如下：假设申请的内存块大小为 rSize，用位操作确定位图索引 si＝fls(rSize)＋1，其中 fls()操作表示取 rSize 左数第1个1的位置. 例如请求30Byte的内存块，$rSize = 30_d = \frac{76543210}{00011110b}$，则 si＝4＋1＝5，对应的固定内存块大小是 2^5 ＝32Byte. 再根据简单对应换算关系：si－3 ＝5－3＝

2,则这个内存块所属的内存分区首地址放在指针数组 p[2]中.若要释放这个内存块,也可根据这个简单对应关系,可以迅速确定要释放内存块所属内存分区对应内存控制块首地址(即对应指针数组元素的值),再据此找到所属内存分区的地址,从而把内存块释放给该内存分区.

对小块内存采取这样的管理方式可以充分利用 μC/OS－Ⅱ原有内存管理算法的优点,并通过增加一个数据结构(即指针数组)和一个简单对应换算关系,就把以前分散的分区通过一个数组串联了起来.针对小块内存的这种内存管理算法,不但保证了内存块分配和回收的执行时间固定,实现了实时性,而且极大地提高了内存的分配和回收的速度(浪费的内存也少),确保了嵌入式系统对内存管理的快速性要求.

3.2 大块内存管理方案

固定大小的内存分配方式对小块内存申请是合适的,但对大块内存(＞1KB)申请则不合适.因为采用固定大小的内存分配方案其内存利用率不高,固定内存块越大,内存利用率越低,因此对大块内存申请应考虑用另外的分配方案.

文中对大块内存申请按实际申请内存大小进行分配[7].按实际大小的内存方式,在内存利用率方面的优越性无疑是最高的,需要多少内存,就分配多少.但这是以频繁分割内存为代价的,频繁分割内存势必会影响内存的分配速度.空间和时间的矛盾在内存管理中是一对时刻存在、无法避免的矛盾,只能权衡利弊,做出取舍.由于只是在大块内存请求时才采用这种分配方案,考虑它在利用率方面的高效性,对大块内存申请就采用此方案.

对于大块内存分配,具体实现如下:

(1) 定义内存分区的内存控制块的数据结构

```
struct BMemHead
{
void*    BMemAddr;    //内存分区指针
BMemHead*    BMemNPar;    //下一个内存分区
BMemHead*    BMemPPar;    //前一个内存分区
void*    BMemBlkListl;    //内存块指针
INT32U    BMemBlkSize;    //每个内存块大小(字节)
INT32U    BMemNBlks;    //该分区中内存块总数
INT32U    BMemNFree;    //该分区中空闲块数
void*    BMemAddrEnd;    //内存分区结尾指针
INT32U    BMemParSize;    //分区大小
}
```

(2) 定义两个双向链表[8]

一个是自由内存区链表,用于管理内存中待分配的的内存块;另一个是分配内存区链表,用来管理已经分配的内存区.初始化时,还要建立一个未用内存控制块链表,定义这个链表只是便于管理大块内存,不代表任何内存区.

每当应用程序申请一个大块内存时,按照最小适合原则查找自由内存区链表,如果找到的内存块大于申请的内存,则分割内存.如果分割后剩余的内存小于某个值(如 32Byte,这个值在实际产品开发时可以根据具体情况进行调整),那么就不用分割,直接使用;否则就分割,并从未用内存控制块链表中取一个控制头来管理新的内存区,并把这个新内存区加入到分配内存区链表.如果在自由内存区链表中不能找到一个合适的内存块,则需要按实际大小向系统申请.

每当应用程序释放一个大块内存时,先把这块内存从分配链表中删除,再把该内存对应的内存控制块归还给未用内存控制块链表,同时把这块内存按其大小插入到自由内存区链表中相应位置,形成一个按由小到大排列的有序链表.

通过多次建立内存区和释放内存区后,自由内存区可能是一个由多个相互链接在一起的表头进行管理的待分配内存区,分配内存区同样是一个由多个相互链接在一起的表头进行管理的已占用内存区,如图 3 所示.如果这时释放内存区可能会让自由内存区中本来碎片化的内存区的部分连成一片,若确实如此,则合并内存区;否则,仅仅是将新释放的内存区插入到自由内存区供下次分配比较使用.

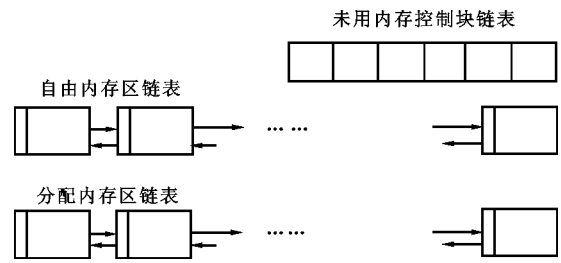

图 3 大块内存分配示意图

大块内存的分配算法不足之处是:当链表逐渐增大时,搜索一个节点的时间会相应变长.但链表是按大小排列的有序链表,而且大多数嵌入式系统中对大块内存请求不多,所以对大块内存分配而言,提高内存利用率才是需要解决的主要矛盾,因此这种分配方案对于大块内存分配是合适的.

4　性能分析

对于小块内存申请，采用固定大小的分配方案，虽有少量内部碎片存在，但简单可靠，内存分配和回收速度都很快，这对于嵌入式系统中大量的内存申请是小块内存来说更注重的是要求时间效率. 根据申请小块内存的大小，计算出它所在内存分区的首地址，然后从空闲链表中获取空闲内存块，时间复杂度为 $O(1)$，释放时为 $O(1)$；

分配大块内存时，根据实际需要来确定申请内存的大小，时间复杂度取决于链表的长度 n，即时间复杂度为 $O(n)$. 虽然大块内存分配和回收的时间相对较长，但其内存利用率很高，而且嵌入式系统往往对大块内存需求量较少，链表的长度 n 不会很大，因而在实际应用中是能满足应用上的时间要求.

另外，内存块大小的分界值大小（假设为 2^k）要综合考虑. 取值过大会造成过多内部内存碎片，而取值过小时，会影响内存的分配效率. 以 SDRAM64MB，μC/OS－Ⅱ，S3C2410(203 MHz)环境下管理 8 MB 动态内存为例，进行动态内存块管理的设备测试，测试数据见表 1.

表 1　内存分配测试数据

k 值	10000 次内存分配时间(ms)	内存利用率(%)
6	119	99
7	120	99
8	118	99
9	119	99
10	118	99
11	117	98
12	116	97

经综合考虑，文中取 $k=10$，即以 1KB 为大小分界值进行分析，实际应用中 k 的取值可以根据具体情况确定.

5　结束语

文中分析了 μC/OS－Ⅱ中内存管理方法的实现机制，在吸取原有内存管理算法优点的基础上，提出了一种新的内存分配算法. 改善了原有系统的性能，使它能较好地应用于需要动态内存分配的场合，特别是对大小内存块都需要的系统更具有优势. 实验结果证明其达到了预期的目的，对相关研究工作有较好的参考价值.

参考文献：

[1] 黄贤英，王越，陈媛. 嵌入式实时系统内存管理策略[J]. 计算机工程与设计，2004，25(10)：1808－1810.

[2] Jean J Labrosse. 嵌入式实时操作系统 μC/OS－Ⅱ[M]. 2 版. 北京：北京航空航天大学出版社，2003：270－282.

[3] 任哲. 嵌入式实时操作系统 μC/OS－Ⅱ原理及应用[M]. 北京：北京航空航天大学出版社，2005：188－203.

[4] 董明峰，谷建华. 嵌入式实时操作系统 eCos 内存分配策略的分析[J]. 微电子学与计算机，2004，21(7)：120－123.

[5] Johnstone M S，Wilson P R. The memory fragmentation problem：solved？[C]//Proceedings of the International Symposium on Memory Management. Coumbia，Canada：[s. n.]，1998：26－36.

[6] 孙晓辉，王劲林，陈晓. 实时系统中的动态内存分配算法[J]. 计算机工程，2008，34(8)：80－84.

[7] 李志军，王铮，王帅. 嵌入式系统的自适应动态内存分配算法[J]. 计算机工程，2007，33(20)：91－93.

[8] 严蔚敏，吴伟民. 数据结构：C 语言版[M]. 北京：清华大学出版社，1997.

作者简介：

李平勇　男，(1974－)，硕士，讲师. 研究方向为嵌入式系统、网络通信.

游　磊　博士研究生，讲师.