第33卷 第20期 计 算 机 工 程 2007年10月
Vol.33 No.20 Computer Engineering October 2007
·软件技术与数据库· 文章编号：1000—3428(2007)20—0091—03 文献标识码：A 中图分类号：TP301.6

嵌入式系统的自适应动态内存分配算法

李志军，王 铮，王 帅
（重庆大学计算机科学与技术学院，重庆 400030）

摘 要：分析了2种常用的动态内存管理算法，基于此提出了一种新的适用与嵌入式系统的动态内存管理方案，在融合了经典算法精髓的同时，通过引入特殊的数据结构，避免了常用算法某些方面的不足，使其更能满足嵌入式系统对内存管理的特殊需求。
关键词：内存管理；伙伴算法；最先匹配算法；动态内存分配算法

Adaptive Dynamic Memory Allocating Arithmetic for Embedded System

LI Zhi-jun, WANG Zheng, WANG Shuai
(College of Computer Science and Technology, Chongqing University, Chongqing 400030)

【**Abstract**】Management of dynamic memory is a very important task of computer science. This paper proposes a new arithmetic of memory management based on two classical arithmetic. By absorbing soul of classical arithmetic and improving their shortcoming, it makes the new arithmetic fit the special requirement of embedded system.
【**Key words**】memory management; buddy arithmetic; first-fit arithmetic; dynamic memory allocating arithmetic

随着各种嵌入式产品的进一步开发和推广，如何在嵌入式系统中更有效地对动态内存进行管理显得越发重要。动态内存管理一直是计算机科学中一个非常重要的课题，尤其是动态内存的分配和释放算法，其主要研究目的是提高动态内存分配和释放的效率，并尽量减少内部碎片和外部碎片，但至今没有很好的解决方案。本文提出了一个新的数据结构及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]。

1 现有传统算法的描述与分析

目前在很多嵌入式应用中，比较常用的动态内存分配和释放技术是最先匹配算法和伙伴算法。

1.1 最先匹配算法[2]

最先匹配算法的基本思路是：定义2个链表，一个是空闲链表，用来管理内存中空闲块，另一个是分配链表，用来管理已经分配的内存块(如图1所示)；出现分配请求时，搜索空闲链表直到找到一个满足内存请求的空闲块，同时相应更新2个链表；内存的释放过程是：先搜索分配链表，找到要释放的内存块，从分配链表中删除，同时空闲链表也相应更新。其缺点是：当链表很大时，搜索一个节点与其长度成正比，而且随着内存碎片的逐渐增多，算法实现的效率也会下降。因为释放一个内存块时，要搜索2个链表，所以释放一个内存块比分配一个内存块花费的时间长。

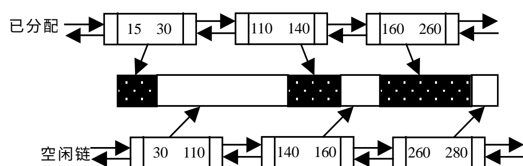

图1 最先匹配算法

分配一个内存块时，先顺序搜索空闲链表，再更新2个链表。顺序搜索在最坏情况下要搜索整个链表，而一般情况下的搜索节点数接近链表总节点数的一半。

定义 $t1$ 为最坏情况下分配一个内存块所需要耗费的时间，$t2$ 为分配一个内存块平均耗费的时间，f 为空闲链表中空闲节点的数，a 为已分配内存块链表的长度。采用遍历的节点数来表示时间。计算结果如下：

$$t1=f\text{；}t2=f/2 \tag{1}$$

由式(1)可知，分配一个内存块，在最坏情况下要扫描整个空闲链表节点，而平均的扫描节点数接近整个空闲节点数的一半。释放一个内存块时，也要顺序搜索分配链表以找到要释放的内存块。由于空闲链表是顺序的，因此必须遍历空闲链表以确定释放的内存块放置到空闲块表中的位置，所以，释放分配块时要涉及到2个链表的搜索。定义 $t3$ 为最坏情况下释放一个内存块的耗费时间，$t4$ 为平均耗费时间，则

$$t3=f+a\text{；}t4=(f+a)/2 \tag{2}$$

从式(1)、式(2)可以看出，无论是分配空间还是释放空间，平均耗费和最坏情况下的耗费都依赖链表的长度。如果在程序的执行过程中周期性地出现2个链表的尺寸短期内特别大，那么程序的运行速度会受到严重影响。

1.2 伙伴算法[3]

伙伴(buddy)系统的基本原理是按照2的幂次方对内存进行分配，以此得到很快的分配速度和回收速度。伙伴系统比那些按大小分配的算法有更多的优点：当一个大小为2^K的块被释放后，存储管理只需要搜索 2^KB大小的块来判定是否需

作者简介：李志军(1980－)，男，硕士研究生，主研方向：嵌入式系统；王 铮，副教授；王 帅，硕士研究生
收稿日期：2006-10-26 **E-mail**：cqlizhijun@126.com

要合并。那些允许内存块以任意形式分割的策略需要搜索所有的空闲块表，因此相比之下，伙伴系统速度更快。但是对于内存利用率，伙伴系统是极其低效的，因为所有的内存请求都要求有 2 的幂次方大小的空间，所以在极端的情况下，内存资源会被浪费近一半。

伙伴系统的内存管理方式不仅对空间的使用存在极大的浪费，在某些情况下还会因资源的浪费导致后继的内存请求得不到满足，进而严重影响程序的正常功能。比如，系统共有 1MB 的内存空间可用，采用伙伴系统进行管理；有 3 个请求：请求 A，260KB；请求 B，260KB；请求 C，10KB。结果如表 1 所示。

表 1　采用伙伴系统进行管理的内存分配

请求	请求空间/KB	分配空间/KB	剩余空间/KB
A	260	512	512
B	260	512	0
C	10	0	0

由表 1 中可以看出，伙伴系统对空间的浪费极其严重，即使在物理内存充足(1MB)的情况下，也会产生请求(570KB)无法满足的情况。

2　本文的内存管理方法

2.1　新数据结构（物理链表）的引入

在最先匹配算法中，分配空间时只需要搜索一个链表，而释放空间时需要搜索 2 个链表，因此，二者的时间耗费不一样，由此引入一种新的数据结构——物理块链表（图 2 所示），用以连接所有物理位置相临的内存块（空闲块和占用块）。物理链表的引入可以极大地节约释放已分配块和合并相临块的时间，因为它只须在物理链表中检查回收块相临 2 个内存块的状态便可决定是否合并。

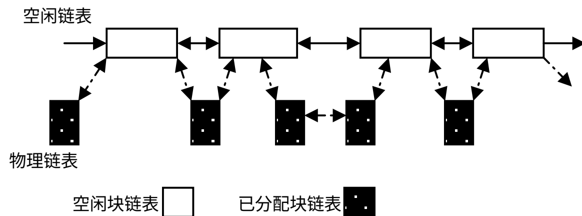

图 2　物理块链表

改进算法的思路是：用一个链表来管理已分配块和空闲块，在分配和释放空间时只须搜索这个表即可。

图 2 中的实线表示空闲块链表，这部分与传统算法中的空闲链表一样，都是双向链表；虚线表示物理块链表。在新结构中，物理块链表和空闲块链表都采用有序的双向链表结构，要释放一个已分配块时，无须像传统算法那样直接搜索已分配块链表，只要通过比较 2 个空闲块节点之间的起始地址就可以找到要释放的分配块所在的子链，从而节约了大量回收及合并相临块的时间。

2.2　大块内存请求的解决方案

根据上述分析，对于大块内存的请求采用按固定大小的分配方式，对内存的利用率是极低的，为了解决这种问题，考虑按实际请求大小来分配内存块。仍采用 1.2 节的例子来分析按实际请求大小分配内存块的方法的利弊，其结果如表 2 所示。

从表 2 中可以清楚地看到，在内存的利用率方面，按实际大小满足请求的内存分配方案比伙伴算法这类以固定大小的内存块高很多，但这是以频繁分割内存块为代价的。空间和时间的矛盾在内存管理中始终存在、无法避免，由于这种分配方案只在大块内存请求时才采用，考虑它在利用率方面的高效性，还是以时间为优先。

表 2　按实际大小满足请求的方法进行的内存分配

请求	请求空间/KB	分配空间/KB	剩余空间/KB
A	260	260	764
B	260	260	504
C	10	10	494

2.3　基于新结构的新算法

buddy 算法中使用多条空闲链表去管理空闲内存块，它把多个固定大小的空闲内存块分别置于不同的空闲块链表中，有利于分配固定大小的内存块，分配和回收速度相当快，并且内存回收也很简单。新算法借鉴 buddy 的这种方法建立了类似的空闲内存块链表，现以 1MB 内存为例，来说明算法的思想。将系统内存区分成 128 个内存仓，小于 512B（作为系统默认大小）的内存仓由固定大小的内存块组成，大于 512B 的内存仓，由小于等于当前空闲块链表基数而又大于前一空闲块链表基数的空闲内存块组成。内存仓从 16B 开始以 8B 的容量递增到 512B，大于 512B 将内存仓分成 1KB, 2KB, 4KB,⋯,128KB,⋯, 512KB, 1 024KB,⋯；同时引入统计域来记录系统对内存的需求情况，以便适时调整内存中的分配。

内存的初始化如图 3 所示，其中，虚线表示物理块链表的连接，实线表示空闲链表的连接。统计域是用来记录相应空闲链表中内存块被应用程序所请求的频率，并根据统计域作为调整空闲块链表的依据。物理块链表被定义为双向链表，用来连接所有物理位置相临的内存块（空闲块和占用块），引入物理链表极大地节约了内存回收后相临块合并的时间，实现了内存块的快速回收，因为它只须在物理链表中检查回收块相临 2 个内存块的状态便可决定是否合并。

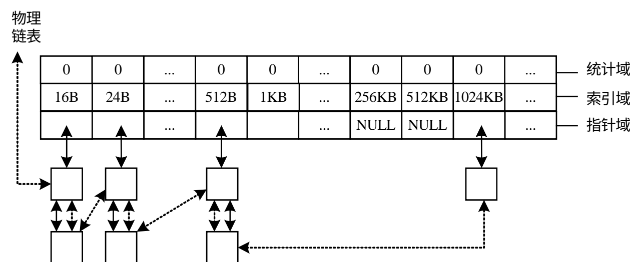

图 3　内存的初始化

自适应动态内存管理算法的原理如下：

初始化时，分配 16B~512B 的空闲链表固定大小的内存块，每个链表 20 块（图 3 中为表示方便已简化），那么计算 {8*(2+3+...+64)*20}/1024≈320KB，1024KB-320KB= 704KB≈700KB，即大于 512B 的链表中只有 1 024KB 链表中有唯一的剩余块 700KB，而统计域值都为 0。

对于 16B~512B 的小块空闲链表，采用分配固定大小内存块的方式来满足分配请求是合适且合理的，因为这样可以避免分割本来就已经很小的内存块，减少内存碎片发生的几率；对于大于 512B 的大块内存请求，不宜采用分配固定大小的内存块的方式，因为这样内存利用率很低，不利于内存有限的嵌入式系统，而应该采用按实际请求大小来满足请求的方法。当系统运行一段时间以后，根据统计域值信息，适当调整空闲块链表。具体调整如下：若统计域为 0，则该固定大小的内存块在这个系统中很少用到或根本不需要，可以将这个空闲链表中的所有内存块合并成一个大的内存块插入到相应链表中；若统计域值很大，说明这种大小的内存块最

常应用，此时即便是物理位置相临的空闲内存块也不进行合并，确保之后可以快速地分配内存；若统计域信息不是很大，说明此类大小的内存块是应用需要但不是最频繁使用的内存块，此时可适当合并物理位置相临的内存块，并将所得的大块内存插入到相应的空闲链表中。

由于 1KB~1 024KB 的大块空闲链表不是固定大小的内存块，因此不仅有根据统计域调节空闲链表的问题，而且还存在内存块的分割问题。根据统计域调节空闲链表可采用上述方法；而内存块的分割不采用固定大小的分割方式，因为固定大小的内存分配方案内存利用率不高，很容易产生大量内存碎片，此时考虑采用按实际大小分配内存的方法，而这势必要分割内存块。但无限制地分割内存块也会导致大量内存碎片，为了避免这种情况，若分割后的空闲内存块小于 16B，那么这次的分配请求不分割内存块，而是将整个内存块分配给请求；当分割后剩余的空闲块大于 16B 时，要插入到相应的空闲链表的相应位，以后有内存请求时，应首先检查刚刚分割剩余的内存块是否满足请求，如果满足请求，首先分配给它。由此把问题归到了一个局部，避免了分割其他的大块内存。

系统运行一段时间后内存状态如图 4 所示（图 4 中的内存块大小只是一种标记，并不表示实际大小）。如图 4 所示，假设系统运行了一段时间后达到了稳定状态，可以发现它对 16B，160B 和 512B 的内存块需求最多也最频繁，这时可以根据统计域的信息将其他固定大小链表中的空闲块合并成大块的内存，并插入到相应的链表中。系统以这种方式运行可以满足嵌入式应用对内存的快速、高效、可靠的需求。

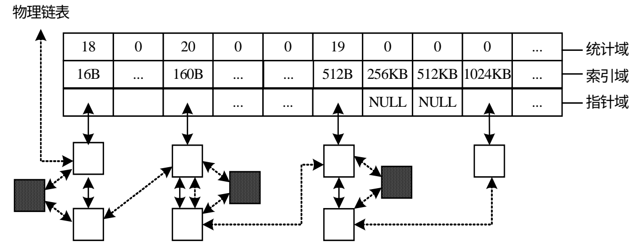

图 4　系统运行一段时间后的情况

3　小结

自适应动态内存分配方法是结合了最先匹配算法和伙伴算法的一种改良算法。新算法中引入了物理链表的概念，为空闲链表和已分配链表确立了某种关系，使释放内存块的时间耗费不依赖于已分配链表的长度，从而提高了算法的实现效率。同时增加了统计域，可以分析出应用对内存块的需求特点，因此，可以采用相应的内存分配方式达到快速、可靠、高效的内存分配，尤其适用于资源有限、没有虚拟存储技术支持的嵌入式应用中。

参考文献

1 曾非一, 桑　楠, 熊光泽. 嵌入式系统内存管理方案[J]. 单片机与嵌入式系统应用, 2005, 1(1).

2 倪西钧, 汤可夫, 吴大为. 一个应用于动态内存管理算法中的数据结构[J]. 兰州理工大学学报, 2004, 30(6).

3 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.

(上接第 90 页)

验：服务器端为一台联想万全 T270，运行 Kylin2.0 服务器操作系统，安装 Apache 2.2.0，对 Apache 源码进行了部分修改，加入了自适应调节机制。客户端由 2 台运行 Specweb99 兼容机模拟客户请求。客户端与服务器用 100Mb/s 以太网相连。图 4 显示了实验结果，其中，横轴为控制间隔个数，纵轴 ΔQoS 为当前服务器的服务质量与期望值的偏差，由响应时间、拒绝概率、吞吐量三者与其期望值的偏差加权求和得到。

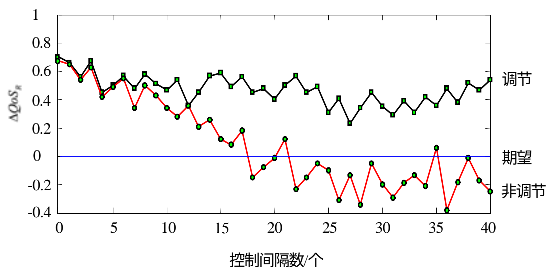

图 4　加入自适应调节机制的服务质量对比

当 ΔQoS 为正时，说明当前服务质量值满足 SLA，ΔQoS 值越大，则服务质量越好；当 ΔQoS 为负时，说明 ΔQoS_R，ΔQoS_P，ΔQoS_X 中至少有一项不满足 SLA；数据显示没有加入自适应调节机制的 Apache 服务器在工作负载到达其峰值时，ΔQoS 为负数，说明 ΔQoS_R，ΔQoS_P，ΔQoS_X 中至少有一项不满足 SLA，而加入自适应调节机制的服务 ΔQoS 始终保持为正，说明该机制能够根据工作负载的变化，自适应地完成调节任务，保证 Apache 服务器的性能及服务质量一直保持在较好水平。

4　结束语

本文介绍了基于实时监控的 Apache 自适应调节机制，给出了模型结构及实现方法，这种自适应调节机制运用分析性能模型结合组合搜索技术，根据负载变化，自适应地完成调节任务；克服了人工性能管理的缺点，实时地调整和改善 Apache 服务器的性能及服务质量，使 Apache 服务器系统一直保持在较佳状态。

参考文献

1 Menascé, D A, Almeida V A F. Scaling for E-business: Technologies, Models, Performance, and Capacity Planning[M]. [S. l.]: Prentice Hall, 2000.

2 Menascé, D A, Almeida V A F. Capacity Planning for Web Services: Metrics, Models, and Methods[M]. [S. l.]: Prentice Hall, 2002.

3 Menascé, D A, Bennani M. On the Use of Performance Models to Design Self-managing Computer System[C]//Proc. of Computer Measurement Group Conf.. 2003.

4 Kleinrock L. Queuing Systems[M]. [S. l.]: John Wiley, 1975.

5 Menascé D A, Almeida V A F, Dowdy L W. Capacity Planning and Performance Modeling: From Mainframes to Client-server Systems [M]. [S. l.]: Prentice Hall, 1994.