第35卷 第20期 计 算 机 工 程 2009年10月
Vol.35 No.20 Computer Engineering October 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)20—0264—03 文献标识码：A 中图分类号：TP393

嵌入式实时动态内存管理机制

顾胜元[1]，杨 丹[1]，黄海伦[2]
(1. 重庆大学软件工程学院，重庆 400044；2. 中兴通讯股份有限公司，深圳 518004)

摘 要：长时间持续运行的通信设备必须满足实时性、可靠性和高效性的需求，针对通信领域的该特点，提出一种嵌入式实时动态内存管理机制。该机制对嵌入式系统有限的内存资源进行统一再分配，为频繁申请和释放内存块的应用分配独立的内存空间。实验结果表明，该机制能实时地提高动态内存管理效率，减少内存碎片，保证系统的健壮性，还提供了内存越界和内存泄漏的检测手段。
关键词：动态内存管理；实时性；分配

Embedded Real-time Dynamic Memory Management Mechanism

GU Sheng-yuan[1], YANG Dan[1], HUANG Hai-lun[2]
(1. School of Software Engneering, Chongqing University, Chongqing 400044;
2. Zhongxing Telecommunication Equipment Co. Ltd., Shenzhen 518004)

【Abstract】The communication equipments running continuously in a long time must meet the needs of real-time, reliability and efficiency, this paper proposes embedded real-time dynamic memory management mechanism aiming at the feature in communications. It can do a unified redistribution with the limited memory resources in embedded systems, distribute an independent memory space for applications which allocate and free memory frequently. Experimental result shows that the mechanism can improve the dynamic memory management efficiency in real-time, reduce memory fragments, ensure the robustness of the system, and provide a method for detecting memory overrun and memory leak.
【Key words】dynamic memory management; real-time; distribution

1 概述

在嵌入式领域中，内存管理不像传统操作系统可以使用虚拟地址映射，从实时性需求考虑一般都采用扁平的物理地址访问。以 Wind River 的 vxWorks 为例，系统中只有一个内存分区，即系统分区，它以一个双向链表管理系统空闲块，采用 First-Fit[1]算法分配内存，释放的内存被聚合形成更大的空闲块。

First Fit 算法会引起 2 个问题：(1)容易导致系统中出现大量外部内存碎片[2]，降低内存使用效率；(2)分配内存的搜索时间和已分配的内存块个数成 $O(N)$的关系，内存分配实时性得不到保证。Wind River 的 pSOS 系统采用的 Bitmapped[1]算法也存在类似问题。

在通信领域，尤其在 SDH 和 WDM 网络中，由于通信量大，在以消息为驱动的基础上释放、分配内存空间非常频繁，因此内存分配回收的快慢对整体性能起着很大的作用，综合通信领域的特殊需求，First Fit 算法以及 Bitmapped 算法都不适合本文的系统。

因此，提出基于 Memory Pool[3]算法和 First Fit 算法的动态内存管理机制。

2 原理

本文提出的内存管理机制，为防止系统运行引起的内存碎片，必须先向操作系统申请一块大的内存区，但不能申请全部系统内存，其原因是操作系统本身需要一些空间来维持自己的运作。分配出来的内存空间称为模块内存。

内存地址自低向高将区域划分为 3 个部分：(1)控制信息区域，用于存放内存的管理信息，用于每一个分配和未分配的内存块管理；(2)动态内存区域，用于大内存的申请管理，该区域的内存被均分为 u 个单元，每个单元大小为 2^v Byte ($v \geq 5$)，用户根据需求配置 u, v 参数，每次内存申请被分配出去的若干个单元为动态内存块；(3)池集内存块区域，用于小内存的申请管理，该区域被划分为 n 个内存池 p_i ($0 \leq i \leq n-1$)，内存池 p_i 有 m_i ($0 \leq i \leq n-1$)个单元，每个单元大小为 2^{k_i} Byte($k_i \geq 3$, $0 \leq i \leq n-1$)，称作池集内存块，用户可根据需求配置 n, m_i, 2^{k_i} 参数，每次池集内存申请只会分配一个池集内存单元。

模块内存分布如图 1 所示。

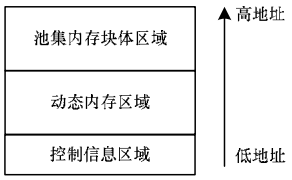

图 1 模块内存分布

模块内存初始化时把池集内存区和动态内存区都初始化为 0x7E，用于内存统计和越界检测。

另外，定义 3 个结构体指向该内存块的不同区域，图 2、图 3、图 4 分别描述了模块控制信息结构、动态内存控制信息结构、池集内存控制信息结构，用于管理模块内存。

基金项目：国家“863”计划基金资助项目“面向通信行业的嵌入式软件开发平台”(2002AA1Z2306)
作者简介：顾胜元(1982－)，男，硕士，主研方向：软件工程，嵌入式操作系统；杨 丹，教授、博士生导师；黄海伦，硕士
收稿日期：2009-04-22 **E-mail：**gushengyuan2002@163.com

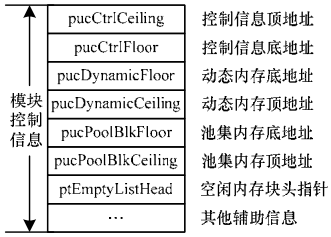

图 2　模块控制信息结构

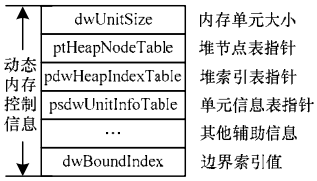

图 3　动态内存控制信息结构

图 4　池集内存控制信息结构

无论是池集内存块还是动态内存块，本文统称为内存块体，所有的内存块体都通过内存块头来管理，内存块头结构和内存块体结构关系如图 5 所示。

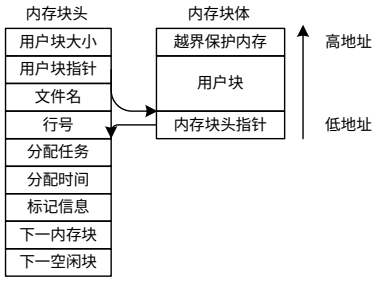

图 5　内存块头结构和内存块体结构关系

由图 5 可知，内存块头中的用户块指针指向内存块体的用户块，用户块才是真正被用户使用的内存区域。用户块前有一小块“ 越界保护内存 ”，用于防止内存的写越界并提供写越界检测。用户块后有一个指针域“ 内存块头指针 ”，该指针回指向内存块头，用于内存释放。

分配内存时，会在内存块头中记录内存分配的文件名、行号、分配任务、分配时间等统计信息。

2.1　控制信息区域

控制信息区域如图 6 所示。

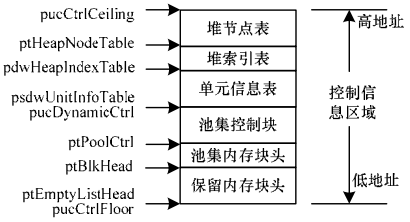

图 6　控制信息域结构

图 2~图 4 中不同指针域指向控制信息区的不同位置。图 6 的控制信息区域存储了管理整个内存模块的信息，用于所有内存块头和内存块体的管理。其中，控制信息域底部存放的是“ 保留内存块头 ”，用于动态内存块申请，数量由用户配置；“ 池集内存块头 ”用于池集内存块申请，数量为 $\sum_{i=0}^{n-1} m_i$ ；“ 池集控制块 ”用于池集内存块区域内存池管理；“ 单元信息表 ”用于管理动态内存单元；“ 堆索引表 ”用于映射堆节点和单元信息表之间的关系；“ 堆节点表 ”用于管理动态内存单元的有效连续内存单元，采用最大堆进行管理。

2.2　动态内存区域

如图 6 所示，通过“ 单元信息表 ”、“ 堆索引表 ”、“ 堆节点表 ” 3 个数据区来管理动态内存区域。动态内存“ 单元信息表 ”保存了每个动态内存单元的使用情况，有 3 类值，如表 1 所示。

表 1　单元信息表值类型

类型	含义
>0	空闲内存块的头内存单元或尾内存单元，值为内存单元数目
=0	已分配或者空闲内存块的其中一块，且不是头或尾内存单元
<0	已分配的动态内存块中的头或尾内存单元，绝对值为内存单元数目

“ 堆节点表 ”保存了动态内存单元的空闲块的信息，空闲块信息包括：空闲块的头内存单元号，空闲块的内存单元数目。然后建立一棵完全二叉树，树中的每一个节点就是一个空闲内存块信息，通过堆算法来查找、插入和删除树中的节点(空闲块信息)。

“ 堆索引表 ”在内存单元信息表和堆节点表之间建立关联，通过该表可以找到内存单元对应的堆节点索引。堆节点索引表的表项数目等于内存单元总数，表项即为内存单元号。表项的值即为空闲内存块对应的堆节点在完全二叉树中的索引。

三者的关系如图 7 所示，例如共有 u 个动态内存单元，因此，“ 单元信息表 ”和“ 堆节点索引表 ”都有 u 项，而堆节点只需 $u/2$ 项即可(最多有 $u/2$ 项)。

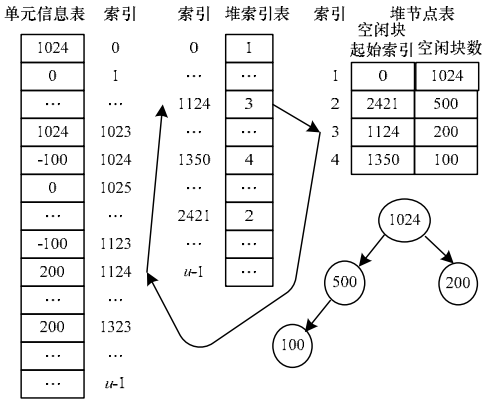

图 7　动态内存块管理结构

如上图中单元信息表可知，0 号~1024 号内存单元以及 1124 号~1323 号，1350 号~1449 号(未画出)，2421 号~2920 号(未画出)内存单元分别是一个连续的空闲块，把这 4 个内存块建立成最大堆后如图 7 右下角所示，该堆以数组方式存放在“ 堆节点表 ”中；“ 堆节点表 ”存储了“ 单元信息表 ”和“ 堆索引表 ”的映射关系。从堆节点表可知，1124 号内存单元是 3 号节点表示的空闲块起始索引单元，通过 3 号节点可知，该空闲块有 200 个内存单元。动态空闲内存块的分配和释放，是对堆节点的调整以及对单元信息表和堆索引表的管理。

2.3　池集内存区域

每一个内存池都由一个内存池控制块管理，所有的内存

池控制块由一个内存池控制块数组管理，如图 6 所示，该数组位于模块内存控制区域中。池集内存控制结构中的 ptPoolCtrl 指针指向内存池控制块数组，如图 7 所示。初始化时，所有的内存池按照内存块的大小升序排序，内存池控制块和内存池的关系如图 8 所示。

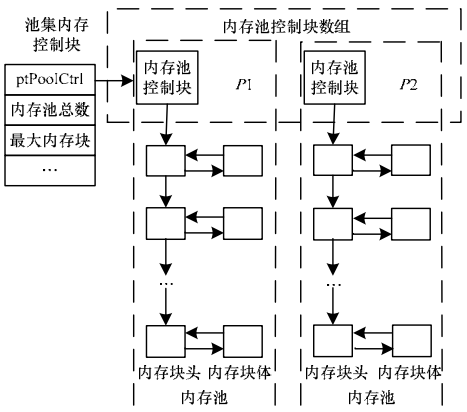

图 8　池集内存关系

图 8 池集中的内存块头和内存块体的关系如图 5 所示。内存池控制块和内存池的关系如图 9 所示。

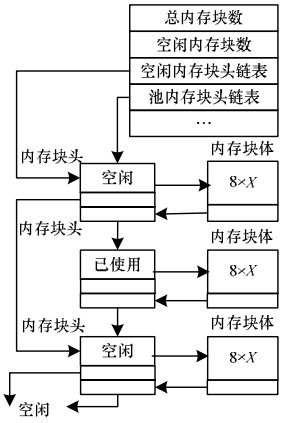

图 9　池集内存分配关系

内存池控制块中保存着内存块的总内存块数、空闲内存块数、空闲内存块头链表、池内存块头链表，每一个内存池 P_i 中的内存块都是通过单向循的池内存块头链表串起来，再通过一个单向空闲内存块头链表管理所有的空闲块。

2.4　内存分配机制

内存申请时，先判断申请的内存是否大于池集内存的最大块。如果不大于，那么搜索池集内存索引表，确定满足需求的内存池。然后从该内存池的空闲内存块头链表获取空闲块，把用户块内存指针返回给用户。否则属于分配动态内存块，先通过堆节点表获取头节点(最大节点)，再判断该节点表示的内存区是否大于申请的内存，如果小于申请的内存，那么返回出错，否则，将申请的内存单元从堆节点中划出来作为一个内存块，按图 8 以堆的方式重新调整堆节点表，以及堆索引表和单元信息表。通过模块控制信息结构的空闲块内存头指针获取一个空闲内存块头，将空闲块内存头指针指向下一个空闲内存块头，然后把该内存块头和内存块按图 5 方式填充相应字段，返回用户内存块指针。

2.5　内存释放机制

释放内存时，判断内存块头标记判断该内存块是属于池集内存还是动态内存。如果是池集内存，则直接删除该内存块头，并标记为空闲，添加到内存池的空闲内存块头链表首部。如果是动态内存，那么先删除该内存块头，标记为空闲，添加到空闲块内存头指针链表首部，并把内存块进行堆合并，同时调整“单元信息表”和“堆索引表”。

2.6　内存故障诊断

本文设计的内存分配机制暂时提供 2 种主要检测手段：内存越界和内存泄露。

释放内存时，通过检测内存块的“越界保护内存”值是否为 0x7E 来确定是否发生了内存写越界。如果发生了，那么通过内存块头中记录的文件名、行号等信息快速定位问题。

内存泄露[4]的主要特征有 2 个：(1)客户申请了内存块而长时间没有释放；(2)大量内存块头是由同一个文件、同一个行号和任务申请的。内存块头中记录了分配时间、文件名、行号、任务 ID 等信息，利用该信息可以分析隐含的内存泄露(并非绝对)，然后快速定位。如 A 文件 B 行代码分配的内存在 10 min 内未释放，或者 A 文件 B 行代码分配的内存块在同一个任务内随着时间递增不断增多且未释放，就有可能是内存泄露。

3　性能分析

采用内存池管理方式以及固定尺寸的动态内存管理方式，避免了外部碎片的产生。虽然有内部碎片，但在每次释放内存的时候可以回收，极大提高了内存利用率。

动态内存单元的尺寸是 2^v Byte($v \geq 5$)，当 v 取值过小时，会影响大内存的分配效率，当 v 取值过大时，会造成过多内部内存碎片。以 SDRAM 256 MB, vxWorks6.4, Freescale MPC8323(333 MHz)环境下管理 16 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 2，当 v 取值为 5~9 时有较高效率。

表 2　内存分配测试数据

v 值	10 000 次内存分配时间/ms	内存利用率/(%)
5	109	99
6	110	99
7	108	99
8	109	99
9	108	99
10	108	98
11	107	97

分配池集内存块时，只要按顺序搜索预先配置好的 n(n 一般在 20 以内)个内存池，然后从空闲内存块头链表中获取内存块，时间复杂度为 $O(1)$，释放时为 $O(1)$；分配动态内存块时，堆节点的个数最多为 $u/2$，分配释放内存时都涉及到堆调整，时间复杂度都为 $O(\mathrm{lb}u)$，以 200 MB 动态内存为例，v 取值为 7，lb$u$ 不超过 21。

vxWorks 采用的 First-Fit 算法效率依赖于链表长度，复杂度为 $O(n)$。因此，本文算法大大提高了时间效率，满足实时性需求。

4　结束语

研究内存管理机制的目的有 2 个：(1)减少对 malloc/ free 的依赖，避免由之带来的内存碎片、时间不确定等因素；(2)增强程序的诊断能力，如写越界保护、内存泄露检测等功能。

本文的机制已在通信领域的 SDH、WDM 网络产品中试用，工程检验表明其很好地满足目前大容量业务的需求，具有很高的推广和实用价值。 (下转第 269 页)