通信设备中内存管理优化

鲁 旭*[1] 马 争[1] 缪 敬[2]

(1. 电子科技大学通信与信息工程学院 成都 610054; 2. 中兴通讯股份有限公司 成都 610041)

【摘要】通过对内存管理的分析，提出了内存优化算法。该算法解决了通信设备中由于大量消息的发送导致内存管理的问题，建立了用户定义的内存管理区域，设计了新的内存管理队列，根据消息块的大小配置内存结构，并给内存块加保护区域，从而提高了内存资源的使用效率，尽可能地杜绝了内存碎片。

关 键 词 操作系统封装层； 应用程序接口； 用户内存区； 页错误

中图分类号 TP 333.1 文献标识码 A

Optimization of Memory Management in Communication Equipment

Lu Xu[1] Ma Zheng[1] Miao Jing[2]

(1. School of Communication and Information Engineering, UEST of China Chengdu 610054;
2. Zhongxing Telecommunication Equipment Ltd. Co. Chengdu 610041)

Abstract This paper brings forward an optimizing memory algorithm by analyzing memory management. In communications equipments, memory management problems are often raised by mass message flow. The algorithm puts forward a series of solutions on these problems, including building user-defined memory management zone, designing optimized management queue, configuring memory structures according to the size of messages and providing protections on memory block. This design improves usage efficiency of memory resources and put an end to memory fragmentations as possible.

Key words operation system subsystem; application program interface; user buffer; page fault

在通信设备，特别是第三代核心网络中，由于系统对效率的要求非常高，系统中有大量的模块间消息传递，所以有大量的内存申请释放操作，而商用嵌入式操作系统的内存管理算法已不能满足现有设备的需求。在嵌入设备中由于应用程序和操作系统是紧耦合，操作系统与应用软件使用的是同一个地址空间，不象一般操作系统各有各的运行空间，导致出现问题后很难跟踪定位，所以必须设计有效的内存管理方法。目前国内外厂商对内存管理的大多数算法都建立在效率严重下降的基础上，在实际产品中很难应用。一般嵌入式系统由于实时性和效率的要求等原因，采用的内存管理方式比较简单，一般不会采用虚拟页面映射，而采用直接物理映射，大多缺少内存的保护措施和内存的错误诊断方法，且很少有内存碎片整理机制[1, 2]。由于嵌入式系统运行环境的特殊性，当出现内存等方面的异常错误时，如果脱离了调试环境就很难定位。故在使用实时系统的过程中，采用自己的内存管理机制是对当前系统的内存管理进行相应的改进措施，使系统更加健壮，能更合理地利用资源[3, 4]。本文基于内存管理优化的设想，在一定条件下杜绝内存碎片，利用软件方法最大程度地提高内存使用效率，并以Vxwork操作系统为例，其他嵌入操作系统与其类似。

2002年9月9日收稿

* 男 26岁 硕士生 主要从事嵌入式操作系统方面的研究

1 内存优化算法

1.1 原理

内存管理模块属于运行支撑系统的一个模块，其主要任务是内存的分配、回收、保护。进程上层任务作为独立的运行实体，其创建、运行、切换和消亡都离不开内存管理模块的参与。另外，为了保护某些重要的系统数据或进程的私有内存空间不被错误的指令所破坏，内存管理模块必须采取一定的保护措施，从而提高系统的可靠性。内存模块位于运行支撑层内，给上层应用和操作系统封装层提供一个统一的内存使用平台[5~7]。

本文基于操作系统的内存管理，在商用操作系统上建立一个操作系统封装层(operation system subsystem，OSS)，内存模块为上层模块提供内存操作的应用程序接口(application program interface，API)调用。

在系统配置中，把所有的自由内存空间设定为用户保留空间，这样，操作系统就不能使用这些空间，上层通过操作系统的内存分配函数就不能申请到空间。根据消息结构的大小，封装层把保留空间预先配置成不同大小的内存块，并用链表串起来。上层应用通过封装层提供的API存取预先定义好的内存，也可对内存模块进行管理，包括申请、释放、回收等[1, 4, 7]。

1.2 用户内存区管理算法的实现

本文模块使用内存管理方法，为防止系统运行引起的内存碎片，须先向操作系统申请一块大的内存区，把大块自由空间定义为用户保留空间，但不能把全部空间都定义为用户保留空间，因为操作系统本身一些协议栈运行也需要一些空间。在Vxworks通过Config.h进行配置时，其他嵌入操作系统也有相类似的配置文件[1, 2]。再将该内存区划分为若干内存池，每个内存池中内存块的大小固定，各种内存块的数量可根据需要配置。为了最大程度上令初始配置与实际使用的用户内存区(UB)数量吻合，在程序中可以增加各种UB峰值使用的统计，以此数据调整各种UB的数量。分配内存时，以固定大小块的方式分配给申请者，内存的管理完全由本模块做。该方法常用于频繁固定大小内存块的申请，所以进行分组时，要注意分组合理，即组间内存块的大小差距要合适，如果太小，分组的意义就不明显，若太大，会造成内存浪费。另外，组的内存块数量要合理，即足够但不冗余(内存块的大小和数量与所支撑运行的应用进程发送的消息数量和大小有关)。

根据实际情况可以将缓冲池分为64，128，256，512，1 024，2 048，4 096，8 192八种，每种的缓冲池用循环队列来进行管理，提供申请内存和释放内存的接口。内存管理提供其调试信息和状态统计信息。申请内存时根据申请内存的大小选择管理队列，从队列的头部摘取一个缓冲区返回指针给申请者，释放时将内存追加到队列的尾部。

每种内存块都有一个内存管理结构，记录下该种内存的使用情况(空闲内存个数、最大利用个数和累计申请该种内存的次数)。内存队列使用一级索引，申请内存的大小直接映射到队列的编号。

图1所示是内存块划分的情况和缓冲池POOL的结构图。图2所示是内存的操作维护算法图，其内存队

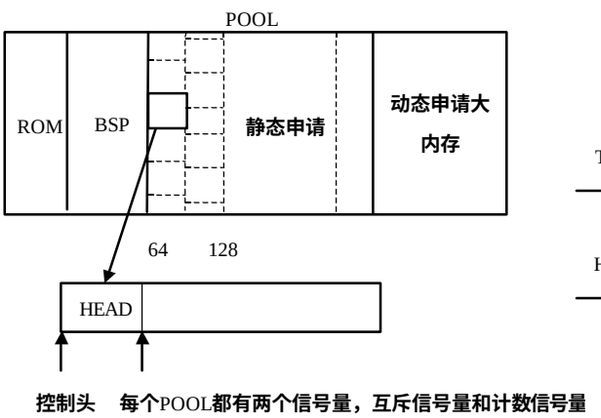

图1 内存划分和缓冲池结构图

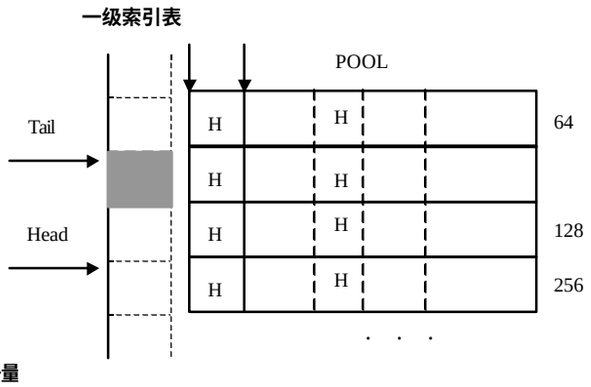

图2 内存的操作维护算法图

列是一个简单的循环队列，申请时从队列头部取一空闲块，归还时放在列尾。由于内存队列是系统每个任务都会申请和归还，故设置了一个信号量来互斥。为了提高效率，内存队列采用二级索引：内存块的大小经过简单的移位得到一级索引值，一级索引值即为内存队列的编号。

申请内存时，根据所需内存数值的移位，得到一级索引值，找到相应的POOL及其对应的内存队列，根据队列的头部，将可用数据块的指针值返回给调用者，其框图如图3所示。

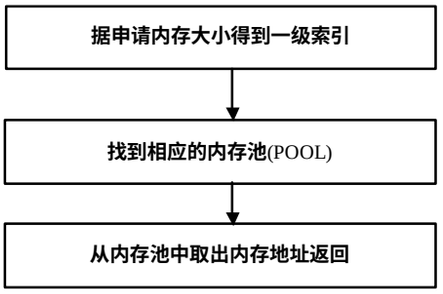

图3　申请内存的算法图

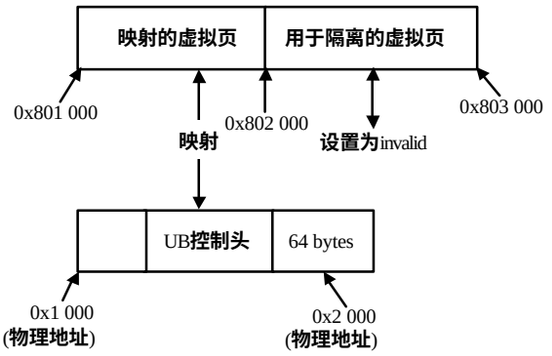

图4　UB保护方式

归还内存时，从HEAD找到相应的缓冲池POOL及其对应的内存队列，将要释放的内存的指针值放入相应内存队列的尾指针。

1.3　UB区的保护

影响嵌入式软件系统稳定性的很大因素是内存问题。内存保护主要是为了隔离进程的地址空间，以达到防止内存非法读写，保证系统稳定地运行。进程作为最小的软件运行实体，其私有数据区、代码区、堆栈、消息队列等对其他进程都不可访问，即具有独占性。另一方面，系统核心数据(MCB、TMCB、PCB，内存映射表)不得被任何应用进程非法篡改，因此内存管理模块必须提供以上内存保护能力，以防止进程对其他进程私有数据和核心数据进行非法访问。因此，在采用UB块的内存管理模式中对于UB块的保护非常关键。

对于UB块的保护，本文提供了图4所示的两种保护方式：

1) 初始化时，在UB的高端加一虚拟页，并让UB向高端靠齐，这是考虑内存地址加加比内存减减的几率高。例如，UB块大小为64 bytes。由图4可知，返回给用户的UB地址是0x801FC0(0x802 000–64)，当用户对该块内存操作时因不慎操作了UB外的地址如0x802 100，则因为该段地址无效，立刻会报页错误异常；

2) 在每个UB块的两端各加8 bytes的保护墙，并给保护墙赋予特殊的码值，在归还此内存块时，通过检查此保护墙的值是否已被改变来得知该内存是否溢出，如果被改变，则该块内存溢出，给出告警信息，并记录是哪个进程调用和该进程的运行状态(如收发消息)，以便定位。如果没有被改变，则该块内存得到了正常使用。这种方法的缺点是事先不能给予保护，只有在归还时才能检测到是否溢出，而这时，相邻的UB块的信息有可能已经被该UB块破坏，如图5所示，返回给用户的是pBuf。

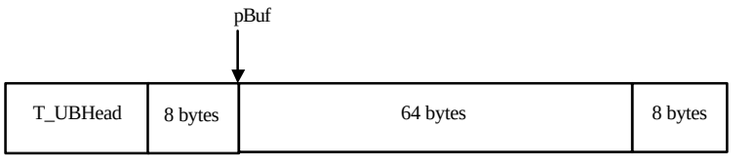

图5　UB保护墙

2　性能测试

设计测试方案模拟内存时，采用UB进行分配释放和不采用UB时的运行状态，统计测试在同等条件下进行相同数量、大小的内存分配释放以检查是否会出现内存碎片，以该指标来量度UB算法的性能。其测试步骤如下：

1) 依照算法原理在VxWorks仿真环境下编译简单例程实现；

2) 对两种不同的方案均事先分配同样大小的内存空间给予分配释放比较。

未采用UB的内存管理常用方案(方案1)调用malloc函数依次循环分配8 K、4 K、1 K，调用free函数释放8 K、1 K，调用malloc函数依次分配1 K、8 K。而采用UB的内存管理方案(方案2)先调用函数InitUB调整各种UB的数量，使8 K、4 K、1 K的数量与方案1中初次使用malloc分配完内存的数量相等，调用GetUB函数依次循环分配8 K、4 K、1 K，调用FreeUB函数释放8 K、1 K，调用GetUB函数依次分配1 K、8 K，分配释放时内存比较如图6所示。

在方案1中，由于malloc分配时采用的首次适配的算法[4]，头部的8 K空间分配了1 K，剩余的7 K和尾部的1 K无法分配给一个8 K，因此出现了内存碎片，malloc is fail!目标机显示：malloc：block too big －8192 in partition 0x3dfe458，但在方案2中采用了UB方案，所以没有出现内存碎片，GetUB is OK!分配释放均正常操作。

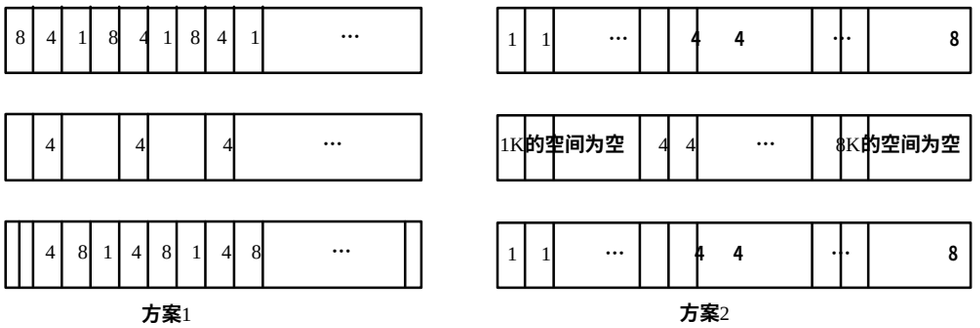

图6 两种不同方案的比较

3 结 束 语

本文提出的算法对于上层频繁使用固定大小内存分区效率的提高十分明显，杜绝了内存碎片的出现，加强了内存的保护。由于上层应用通过该封装层实现对内存的操作，因此当出现错误时不易出现整个系统的崩溃。嵌入式操作系统封装层内存模块的使用，在相当程度上降低了嵌入式软件在内存部分开发的难度和投入。另外，该封装层只是在原有嵌入式操作系统内核的基础上进行封装，因此上层应用软件的编译、调试环境还是原有的环境，为了增强通用性，封装层软件需要增强上层应用软件的调试与检测功能。随着嵌入式操作系统正向标准化、模块化的方向发展，且其操作系统的内核与标准正向类似UNIX和LINUX的方向靠拢，故掌握操作系统内核的核心技术，开发一套完善的系列工具，对于通信设备中嵌入式软件的开发具有很大的促进作用。

参 考 文 献

[1] Jean J L. µc/os – II 源码公开的实时嵌入式操作系统[M]. 北京：中国电力出版社，2001

[2] Galli, D. Distributed Operating Systems: Concepts and Practice[M]. Upper Saddle River, NJ: Prentice Hall, 2000

[3] 孔祥营，柏桂枝. 嵌入式实时操作系统VxWorks及其开发环境Tornado[M]. 北京：中国电力出版社，2002

[4] William S. Operating systems: internals and design principles (fourth edition)[M]. 北京：电子工业出版社，2001

[5] Laplante P A. Real-time systems design and analysis，an engineer’s handbook[C]. Piscataway，New Jersey: IEEE Computer Society Press, 1992

[6] Allworth S T. Introduction to real-time software design[M]. New York: Springer-Verlag, 1981

[7] Ganssle J G. The art of programming embedded systems[M]. San Diego: Academic Press, 1992

编 辑 徐培红