一种嵌入式安全内存管理的设计与实现

赵跃华， 蔡贵贤， 黄卫菊
(江苏大学 计算机科学与通信工程学院，江苏 镇江 212013)

摘 要：针对电力应用系统对嵌入式操作系统的实时性和安全性要求，以目前流行的嵌入式操作系统 uC/OS-II 内核为基础，巧妙地应用伙伴算法增强了内存管理模块的功能，并实现了存取控制和审计跟踪等安全功能，同时解决了内存空间的泄漏问题。经测试，本设计能很好的满足电力应用的要求。

关键词：嵌入式操作系统；内存管理；安全性设计；伙伴算法；uC/OS-II
中图法分类号：TP316.2 **文献标识码**：A **文章编号**：1000-7024 (2006) 16-3092-04

Design and implementation of embedded and secure memory management

ZHAO Yue-hua, CAI Gui-xian, HUANG Wei-ju
(School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China)

Abstract: According to the time and security requirement of electric application on embedded operating system, the buddy system to enhance the capacity of memory management skillfully is applied, which is based on the uC/OS-II. At the same time, the security policies are realized, such as, the secure access to memory and audit come true, and the problem of memory leaking is resolved. By testing, it is well adaptive in electric application.

Key words: embedded operating system; memory management; security design; buddy system; uC/OS-II

0 引 言

嵌入式系统在电力系统中的应用有着悠久的历史，随着 Internet 技术的飞速发展，电力嵌入式系统连接 Internet 必是一大发展趋势。但是，由于 Internet 存在更多的安全性威胁，安全问题也必是一个更加突出的、必须解决的问题。这就要求在设计嵌入式操作系统时，除了考虑实时性等要求外，还必须考虑到它的安全性设计。

目前，比较流行的嵌入式操作系统有 QNX、VxWorks、Window CE、uC/OS-II 等。由于 uC/OS-II 内核具有稳定、简短、源代码公开等特征，所以得到了人们越来越多的青睐。但是其内存管理过于简单，并且没有安全性方面的设计，为了更加适合于在电力系统中得到应用，这部分功能必须增强。

在嵌入式系统中，物理内存空间是除了 CPU 之外最基本、最重要的资源，物理内存空间在系统中的分配和回收效率对于系统的效率起着重要的决定作用。所以如何高效地分配和回收内存空间、如何减少内存碎片带来的资源损失、如何实现内存管理的安全性等，这些都是设计和实现嵌入式操作系统安全内存管理所必须考虑的问题。

1 uC/OS-II 内存管理分析

在 uC/OS-II 中，操作系统把内存按分区来管理。每个分区中包含整数个大小相同的内存块，以块作为分配和回收的单位。系统为每个内存分区分配一个内存控制块结构，用结构体 OS-MEM 表示。OS-MEM 中登记了分区起始地址、分区中块大小、数量等信息。应用程序在使用一个分区前，必须先建立该内存分区，从而可以从已建立的合适的分区中申请一个内存块，使用结束后必须归还。

从上面的分析可知，uC/OS-II 的内存管理实现的功能过于简单。首先，当内存分区中没有合适的空间分配时，需要重新建立一个分区或者找一块比较大的内存块分配，这样要么会牺牲系统的实时性，要么会引起内存碎片的增多，这样显然不太适合于电力应用领域的实时性要求。其次，uC/OS-II 的内存管理没有提供存取控制、审计跟踪等安全性措施，显然这不能满足越来越紧迫的安全性要求。最后，对于嵌入式操作系统必须面对的一个严重问题——内存空间泄漏，uC/OS-II 的内存管理对它也是无能为力的。

2 基于伙伴算法的安全内存管理的设计与实现

2.1 伙伴算法

伙伴算法(buddy system)是一种经典的内存分配和回收算法。由于这种算法实现简洁、速度较快，在各种系统中有较广泛的应用。在伙伴算法中，分割 (splitting) 机制用来寻找合适的空闲内存块，而合并(coalescing)机制用于在释放内存块后将

收稿日期：2005-06-02。
作者简介：赵跃华 (1958－)，男，江苏苏州人，博士，教授，研究方向为嵌入式系统、信息安全； 蔡贵贤 (1981－)，男，硕士，研究方向为嵌入式操作系统、信息安全； 黄卫菊 (1977－)，女，硕士，研究方向为嵌入式网络、信息安全。

相邻的伙伴块合成大的内存块。不同的伙伴算法中，采用不同的机制来完成分割和合并工作，其中二元伙伴算法是最常见的一种。除此之外，还有 Finbonacci 伙伴算法、带权伙伴算法和双伙伴算法等。

本文根据嵌入式操作系统的具体情况，巧妙地、高效地应用了二元伙伴算法实现了内存空间的分配和回收。

2.2 内存空间的划分

在嵌入式系统中，内存空间存放的数据千差万别、操作各异，所以根据数据的差异性，对它们进行分类并存放于不同的位置，这样必然更有利于内存空间的管理。当然，这必须对内存空间进行合适的分区。如图 1 所示，本文根据电力系统中存在的数据的特征，把内存空间划分成了 3 个区。

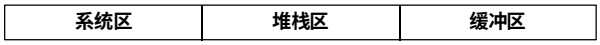

图 1 内存空间分区

其中，系统区主要存放系统程序和应用程序的映象、用户信息、文件系统和内存模块的一些配置信息等，它不再进行小分区划分，在系统初始化时就填入信息，以后改动也相对较小，所以下面就不再介绍此区。堆栈区和缓冲区采用固定分区策略，进行小分区的划分，以便于能够方便快速地处理。虽然这两个区的一些具体处理可能不尽相同，但是对其管理策略是相同的，下面将具体介绍它们的分配和回收管理策略。

2.3 安全内存管理模块分配和回收策略的设计

在内存管理模块中，内存空间的分配和回收是最重要最基本的操作，一种合理的分配回收策略能够使内存管理模块在时间、空间、灵活性等方面达到一个满意的组合。本文综合了时间和空间等方面的要求，并结合电力系统应用的实际情况，采用二元伙伴算法(binary buddy system)作为内存空间的分配和回收算法。

根据伙伴算法的思想，本文把堆栈区划分为 N (N∈Z) 个小分区，每个小分区中含有整数个大小相等的块，n 号分区包含的内存块的大小为 2^n KB。同分区中的块通过 STR_mm_block 结构以双链表链接在一起。

所有的小分区按照包含的内存块的大小，依次连续地排列在从低地址到高地址的内存空间中。每个小分区用一个数据结构 STR_mm_area 表示，并用结构数组进行统一的管理。

在本文所设计的内存管理模块中，各个数据结构和内存单元的关系如图 2 所示。

2.3.1 一些重要数据结构的设计

上述提及两个很重要的数据结构——STR_mm_area 和 STR_mm_block，加上为了解决内存空间泄漏问题而增加了 ARR_mm_pno[63]，将一并介绍。

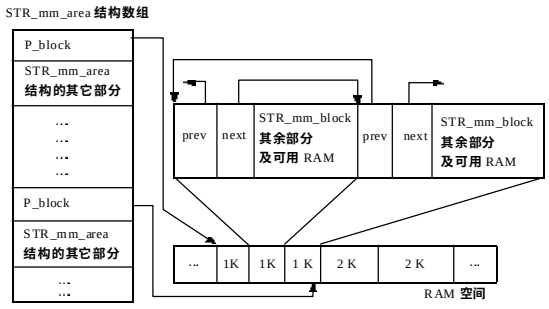

图 2 内存管理组织

(1) STR_mm_area

此数据结构是用来管理内存中的各个小分区，定义如下

```
struct STR_mm_area{
    STR_mm_block *p_block;// 用于指向本分区中的第一数据块
    int totalblock; //本分区中所包含数据块个数
    int emptyblock; //本分区中所包含空闲块个数
    unsigned int *addstart;//本分区的开始地址
                //本变量由用户配置，并在初始化时赋值
    unsigned int *addend; //本分区的结束地址
    int k; //本分区中内存块空间大小的指数值
}; // STR_mm_area
```

(2) STR_mm_block

此数据结构是用来管理内存小分区中的每个数据块，它放在内存块的首部，这是一个非常重要的数据结构，内存管理中许多功能的实现都得依靠其支持。此数据结构的定义如下

```
struct STR_mm_block{
    int k; //意义同 STR_mm_area 结构中的 k
    int k_now;// 登记当前该块的大小的指数值
    STR_mm_block *prev;//指向该块的前一块
    STR_mm_block *next;
int state; // 该块现在的状态，有两种状态，
        //0 表示该块空闲，1 表示该块被占用
int security; // 占用该块的任务的安全级别，当此块空闲时，它的值为-1
int pno;  //表示占用该块的任务的任务号
}; // STR_mm_block
```

(3) static int ARR_mm_pno[63]

该数据结构是用来存放内核中存在任务的任务号，因为内核中每个优先级别每次只能为一个任务所占用，所以可以用数组的下标蕴含标识出任务的优先级。

2.3.2 基于伙伴算法的分配和回收函数的设计

(1) 分配内存块操作：假设任务需要的空间大小是 n KB，那么根据式 (1) 可以求出 k 的值，k 就是最合适的分区号码，也即从 k 号的分区中取得的块是最合适的。

$$k=\begin{cases}0 & \text{当 } n<1 \text{ 时}\\ \lceil \log_2 n \rceil & \text{当 } n\geqslant 1 \text{ 时}\end{cases} \quad (1)$$

若 k 号分区中还有空闲块，则只需直接从中取出一块分配出去，并称 2^k KB 为最合适分配空间。若 k 号分区中已没有空闲块，那么就一直进行 k++操作，直到当前 k 号分区中有空闲块，或者 k>N (N 为包含最大块的分区号) 为止。若 k>N，表示现在内存中已经没有空间可以满足该任务的要求，那么系统就发送一个信号量，把当前任务挂起。若 k≦N，表示在其它分区中找到了满足要求的块，根据伙伴算法的特点，找到的块的大小必是任务最合适分配空间的 2^a 倍 (其中 a≧0，且 a∈Z)。并且本文规定：分配出去的块是取出块的低地址部分。那么就可以把取出的当前块切成大小相等的两块，后面一块标注上必要的信息后就可以直接挂到 k−1 号分区中，为了使得

该块在回收的时候尽可能地与其伙伴块合并，在挂到 k－1 号分区中时，应该尽量挂到链表的尾部。接下来继续对前面的块重复进行上述的切半操作，直到最后切出的块的大小恰好是最合适分配空间为止。该算法的程序流程图如图 3 所示。

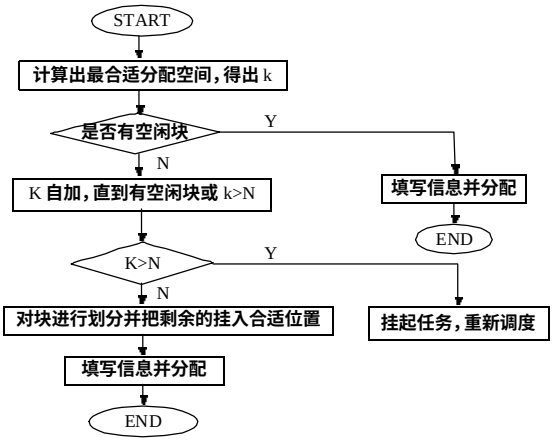

图 3　内存块分配

(2) 返回内存块的操作：返回内存块的操作和分配内存块的操作是息息相关的。在返回内存块时，涉及到伙伴块合并的问题。那么何谓“伙伴”？当两个块之间同时满足块的大小相等、块的地址相邻、两个块是从同一个大块中分离出来这 3 个条件时，称为“伙伴”。

假设当前块的首地址是 p，通过查找 STR_mm_block 可以找出 k 和 k_now，若 k_now<k，说明当前块一定存在“伙伴块”。通过查找 STR_mm_area[k]可以找出分区的首地址 addstart，记 q 为当前块的“伙伴块”的首地址，那么可以通过式 (2) 计算 q。

$$q=\begin{cases}p+2^{k\text{-}now} & 若(p-addstart)\mathrm{MOD}2^{k\text{-}now+1}=0\\ p-2^{k\text{-}now} & 若(p-addstart)\mathrm{MOD}2^{k\text{-}now+1}=2^{k\text{-}now}\end{cases} \quad (2)$$

当任务返回一个存储块的时候，它先查找位于该块头部的 STR_mm_block 结构，比较 k 和 k_now 这两个变量，若相等，说明此返回块本来就位于该分区中，所以直接清除一些信息就可以了。若 k_now < k，可知此块最初不属于此分区，应找出它的伙伴。接着判断它的伙伴块是否为空，若不空则直接修改 STR_mm_block 的信息即可，该块仍处于当前分区中；若为空则合并，接着继续找合并后的新块的伙伴块，重复上述操作，直到 k_now ＝ k 或者不能在进行下一步合并为止，此时应该修改结构 STR_mm_block，并把一些多余的 STR_mm_block 清除掉，最后把合并后的块链接到合适的分区中。该算法的程序流程图如图 4 所示。

2.4　安全内存管理实现所需要的内核支持

为了实现本文所设计的安全内存管理，必须对 uC/OS-II 的内核进行必要的修改。首先，必须在内核控制块中增加 security 这一个变量，内核在创建任务时必须对它赋值，这是实现安全性的关键。其次，uC/OS-II 的 TCB 中保留了一个 OSTCBId 的字段，此处应该利用该字段作为任务号，在创建或删除一个任务时，必须在全局变量数组 ARR_mm_pno[]中的合适位置登记或删除该任务号。另外，uC/OS-II 在调用函数 OSTaskCreate()或 OSTaskCreateExt()创建一个任务时，已经事先定义了一个存储空间，此处应该把存储空间的定义放在这两个函数的

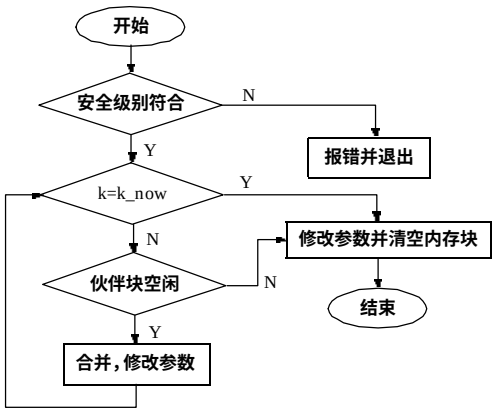

图 4　内存块回收流程

内部，而函数传入参数 pbos 改为 n，表示申请存储空间的大小。

3　安全性的实现以及内存空间泄漏问题的解决

3.1　安全性问题的实现

安全性方面的设计是嵌入式操作系统的一个发展方向，在本文的内存管理的设计中，提供安全存取控制、安全审计跟踪功能、数据本身的安全性等安全方面的设计，以便于对整个嵌入式操作系统的安全性设计提供必要的支持。

3.1.1　安全存取控制的实现

根据电力系统的实际情况，本文采用基于主体权限的存取控制方式。并且基于效率和实际情况的考虑，本文对存储控制矩阵进行了大大的简化，即规定同样的安全级别的任务对所有客体的操作权限是一致的，这样，系统只要为所有相同安全级别的用户配置一个权能列表 (CL) 即可。

在本设计中，安全的主体是系统任务和用户任务，本文把主体的安全级别分为 3 级，分别为 1 级、2 级和 3 级。其中，3 级是最高级别，此种主体是超级任务，拥有一切的操作权限；2 级是操作任务，拥有除了设置安全级别等安全性操作之外的一切权限；1 级是普通任务，只拥有对部分数据的浏览权限。

从上面的设计和分析可以看出，本文是通过下面方法实现安全存储访问控制的：①系统初始化时会根据用户的需求装入一个默认的 CL。系统正常运行时，安全级别为 3 的任务可以对 CL 进行修改。②在该模块的一些重要的数据结构中，增加了安全性字段(security)，为算法设计时实现安全存取控制提供了保证。③在算法设计时，加入了基于安全性的存取判断，当某个主体对某个客体进行存取访问时，应该根据安全级别查找 CL，以确定是否允许该操作，从而实现安全存取控制。具体过程如图 5 所示。

3.1.2　安全审计跟踪功能的实现

在内存管理模块中，会在合适关键的操作中加入事件产生器 (例如，报错处理时，或对一些关键数据的操作时)，对用户进行的一些重要的操作按照预先定义好的标准进行记录。在合适的时刻，系统创建一个审计任务，该任务按照某种规则对这些记录进行分析，从而产生警告信息，并进行相应的处理，同时把这些警告信息存入系统的审计库中。

3.1.3　数据本身的安全性设计

对于那些敏感的数据，在内存中划分一个特殊的区域进

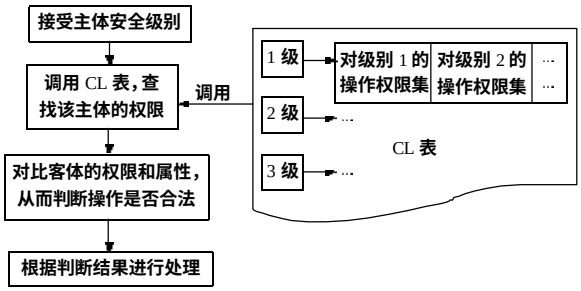

图 5　安全存储控制流程

行加密存储。同时,为了保证数据的完整性,在这些数据加上了必要的纠错信息,并进行了必要的备份。

本文采用 DES 算法对敏感的数据进行加密和解密,并且加上 CRC 循环校验码用于校错,同时在内存中保存数据的备份,用于纠错。为了防止备份数据也遭到破坏,在内存中必须同时保存 4 份数据,这样可以保证内存中任何时刻必有两份数据是一致的。当对这些数据进行修改时,必须同时对其备份数据也进行修改。

这些操作无疑会牺牲系统实时性、浪费内存空间。但是,在电力系统中,这部分数据的量比较小、操作也不频繁,所以不会对系统的整体实时性和内存空间的利用率带来太大的影响,但是却进一步增强了系统的安全性。

安全性的设计涉及到方方面面,限于篇幅,本文不做进一步的展开讨论。

3.2　内存空间泄漏问题的解决

所谓的"内存空间泄漏"指的是,当任务结束后,由于某种原因而没有对占用的内存空间进行释放。这样,这些空间就没有办法再次进行回收利用,久而久之,对系统资源要求很高嵌入式系统所造成的不良影响是可想而知的。

在本文的设计中,当系统空闲时,可以创建一个任务,用来遍历缓冲区和堆栈区,对于非空闲的内存块,查找内存块中的 STR_mm_block 结构和全局表 ARR_mm_pno[],以便确认占用该块的任务是否还存在,从而可以判断是否应该强制释放空间,以此实现对"内存空间泄漏"问题的解决。

4　结束语

利用基于 ARM4510BCPU 芯片和 HY57V641620DRAM 芯片的硬件平台,本文所提出的安全内存管理的设计方案已基本实现。并且,作者修改了本实验室已实现的智能脱扣器项目的软件,并把它加载到修改后的嵌入式操作系统上进行初步的测试。测试结果表明:系统的实时性和安全性均能满足要求。在本论文的基础上,作者将对系统进一步完善和优化,使其稳定性和实时性进一步增强,以便更适合于实际的电力应用领域。

参考文献:

[1] John Carbome.Efficient memory protect for embedded system [EB/OL].*http://www.rtcmagazine.com/home/printthis.php*.

[2] 毛德操,胡希明.Linux 内核源代码情景分析(上) [M].杭州:浙江大学出版社, 2001.

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

[4] 严蔚敏,吴伟民.数据结构(C 语言版)[M].北京:清华大学出版社, 2000.

[5] 沈勇,王志平.对伙伴算法内存管理的讨论[J].计算机与数字工程, 2004,32(3):40-43.

[6] 李小勤,孙玉芳.Linux 内存管理机制的分析与研究[J].计算机科学, 2002,29:18-20.

(上接第 3091 页)

v_tempc、v_tempi 在前面已经定义为 John 和 112),这个 SELECT 语句变成了 SELECT name, birthday, description INTO v_name, v_date, v_description FROM emp WHERE name = 'John' and empno=112;

然后把新的语句作为一个字符串交由 GKD-Base 的 SQL 引擎进行语法检查。以上一条 SQL 语句为例,完成对它的格式转换之后,首先调用 XPI 函数为这条 SQL 语句打开一个游标;然后再调用 XPI 函数对其进行分析,检查这条语句中的变量与对应的列的类型、规模、精度等是否匹配;检查通过后,由 XPI 函数对整条 SQL 语句进行语法分析,保存分析结果或由 XPI 函数给出错误信息;最后调用 XPI 函数关闭打开的游标,对 SQL 语句解析完毕。

4　结束语

本文讨论了 GKD-ESQL 系统的设计与实现技术。该系统由两部分组成,一部分是预编译部分,另一部分是运行库部分。预编译部分处理含有 SQL 语句的源程序,识别其中的 SQL 语句,并将其转换成相应的 C 语言函数,ESQL 运行库就是实现这些函数的功能,从而通过 C 语言编译器的处理,就可以将嵌入式 SQL 语句编译成可执行代码,并在 GKD-Base 数据库管理系统中执行。今后可以在此基础上扩充一些功能,如完善错误处理机制,对指示符变量的支持等。

参考文献:

[1] Hector Garcia-Molina, Jeffrey D Ullman, Jennifer Widom. Data base system implementation [M]. *USA: Prentice Hall*, 2000.

[2] Date C J. An introduction to database systems [M]. 7*th Edition*. *USA: Addison Wesley*, 2000.

[3] The PostgreSQL Global Development Group. PostgreSQL8.0 developer's guide [M].*USA: Postgresql.org*, 2005.

[4] 唐扬,熊伟,赵锋锐,等.数据库功能扩展技术研究与实现[A].NDBC2004 论文集[C].厦门:计算机科学, 2004.

[5] John R Levine, Tony Mason, Doug Brown. Lex and Yacc [M]. *USA: O'Reilly and Associates Inc*, 2003.

[6] 陈火旺,刘春林.程序设计语言编译原理[M].第 3 版.北京:国防工业出版社, 2000.