嵌入式操作系统内存管理研究

田令平
(华东师范大学软件学院,上海 200062)

摘要:嵌入式操作系统的内存管理有其独特之处。基于开发嵌入式操作系统内存管理模块的实际经验,研究了嵌入式操作系统的内存管理方法,分析了本领域当前的一些常用技术,指出了它们的特点及适用条件,总结了一些有效的设计实现原则,针对目前单一内存管理策略在移植性上的不足,探讨了一种实用的可配置的内存管理方案。

关键词:嵌入式系统;操作系统;内存管理
中图分类号:TP316 文献标识码:A 文章编号:1009-3044(2006)11-0161-03

Research on Memory Management in Embedded Operating System

TIAN Ling-ping
(Software Engineering Institute,East China Normal University,Shanghai 200062,China)

Abstract:The memory management in embedded operating system has its own characteristics.Based on author's development experience of memory management module in embedded operating system,researches methods of memory management in embedded operating system,analyses some currently common-used technics of this field,points out their characteristics and their using conditions,concludes some effective principles of their design and implementation,and discusses a practical and configurable scheme of memory management.

Key words:embedded system;operating system;memory management

1 引言

内存管理是操作系统的中心任务之一,内存管理模块通常是操作系统内核的一部分。其主要任务是组织内存以容纳内核和各待执行进程,跟踪当前内存使用状况,在需要时为进程分配内存,其使用完毕后释放并回收内存[1]。内存管理与操作系统和底层硬件体系结构密切相关,不同平台上的内存管理策略往往大不相同。

较之普通桌面型操作系统,嵌入式操作系统中的内存管理有其独特之处:嵌入式系统受成本、设备体积和电池技术等因素制约,内存容量一般相当有限,需要高效地使用;嵌入式系统一般都具有实时需求,至少是软实时的,内存分配、回收必须迅速,时间应可预测;很多嵌入式系统事关重大,必须安全地运行,为避免灾难性后果,内存管理必须十分可靠。

2 内存管理策略概述

内存管理策略大致可分为静态分配和动态分配两大类。

静态分配的可用内存大小在编译时确定,系统初始化阶段预先分配好固定数量的内存,用于存放所有需使用的对象和数据结构。程序运行过程中能且只能使用这些内存。

动态分配的内存在程序运行时根据需要向系统申请后获得,只有在需要的时候,才分配或归还。

由于嵌入式系统的特殊性,开发人员必须在编译内核和开发应用时都参与系统的内存管理。

3 内存管理模块策略及实现考量

3.1 简单内存模型

设计之初,最简单直接的一种技术其实就是"不管理",即所有静态分配在栈中的、动态分配在堆中的变量和数据结构,这二者的分配、释放,全交给编译器、链接程序和运行时库去完成,开发者只需负责合理设置系统栈、堆及任务栈。该模型的一例如图1。

这种方法一般用于操作系统原型的设计,易于配置的、或内存相对宽裕且实时性要求不高的系统的初始设计。

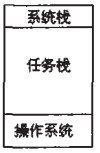

图1 一种简单内存模型

3.2 静态方式

为简单、特殊考虑或安全计,可以只使用静态技术。

静态内存分配在程序执行前进行,效率较高;编译时可精确预估系统内存需求状况;所有分配动作所需时间固定且很小;分配的内存一般不归还,外部碎片很少;程序易测试,可靠性高。这些优点使得静态分配尤其适用于实时应用和高安全系统。

另一方面,静态分配必须考虑可能发生的最坏情况,因此往往分配比实际用量更多的内存。预先分配内存也增加了系统的启动时间。使用静态分配,常需限制系统中同时容纳的进程数,结束旧任务之后才能开始新任务。为限制容量,必须指定程序将存储的最大数据量。这些均会降低系统的可用性[2]。要求在程序执行前就知道所需的内存类型和数量,显然缺少灵活性。

比如,在允许执行动态分配的系统中,任务栈大小本可简单地通过一个参数直接传递进任务的构造函数:

```
Task::Task(void (*pfunction)(), Int32U stackSize, …)
{pStack=new Int32U(stackSize);
//…}
```

但在只采用静态分配的内存管理方式中,不仅操作系统内核本身,用户进程也必须全部采用静态分配。所以,只能通过以下几种方式设置运行前已指定了大小的任务栈:由用户传递一个静态分配的数组首地址参数给任务构造函数;使用模板参数;继承通用任务类,实现自己的特定任务子类,在子类构造函数中给予其静态分配的栈。用户传递一个静态分配的数组首地址参数方式示例如下:

```
Task ::Task(void (*pfunction)(), OS_STK* pts, Priority prio,…);
```

收稿日期:2006-01-26
作者简介:田令平(1976-),男,江苏南京人,硕士研究生,研究方向为嵌入式系统。

OS_STACK task1Stack[task1StackSize];

Task task1(&task1Entry, &task1Stack, Task1Prio…);

静态分配也使用前述简单内存模型，只不过不进行任何动态操作。

3.3 动态方式

大部分面向过程和面向对象语言都提供在堆中创建和回收对象的动态管理机制，并在缺省状态下使用。

静态分配注重效率和安全，动态分配则偏重于灵活性。它避免分配不被使用的内存，降低内存总体需求；改善设计质量，使系统更易编写、扩展和维护；运行时间分配内存减少了系统的启动时间。

当然，灵活性也需付出代价：动态内存管理需要额外区块头部的开销；内存分配和归还需花费处理器时间，典型情况和最坏情况下的内存需求难以估计；内存耗尽的处理会增加代码的复杂度；易造成内存碎片，增加内存的额外开销和不可预测性。

动态分配的内存必须由操作系统管理[1]。通常使用的是链表法，它维持一个已分配和空闲的内存段的链表。

段是进程或是两个进程间的空洞。段链表通常按照地址排序，在进程结束或被换出时更新十分简单、直接。使用双链表结构更易于找到上一个表项以检查是否可以合并。

进程和空洞按照地址顺序存放在链表中时，在系统知道要分配的内存大小的前提下，可用于为新创建和换进的进程分配空间的算法中，首次适配不仅是最简单的，而且是最好和最快的[3]。首次适配算法从头开始扫描存储器并选择大小足够的第一个可用块。

把进程和空洞放在不同链表中，能提高这几个算法的速度，但会提高复杂度和使内存释放速度变慢，因为被释放的内存段必须从进程链表中删除并插入空洞链表。

将空洞按大小排序将导致查看是否可以合并邻接块的操作非常费时。但若不作合并，内存将很快充满大量的、进程无法使用的碎片。所以，空洞列表一般按照内存地址排列，需要分配内存时，用首次适配算法对空洞列表进行搜索，以找出一个足够大的空洞并装入之[3]。

在必须使用动态方式的系统中，也应同时运用静态方式，把所有可以事先确定的内存用量都预先分配好，以尽量减少动态分配和回收带来的不确定性。

3.3.1 碎片

碎片是动态内存分配、尤其是直接在系统堆中分配的不分区方式的主要问题。有两种碎片，一种是数据结构未用完它所分配到的内存块而产生的“内部碎片”，另一种是两个已分配数据块之间的内存由于太小而无法用于存储任何东西造成的“外部碎片”。碎片很难消除：如果按实际需要分配大小不一的区块，那就容易造成外部碎片；如果为消除这种情况而令每个区块大小相同（取各需求的最大值），那就造成因每个区块未被完全使用而形成的内部碎片。

以下是一些减少碎片的措施：为避免大对象和小对象到处散布而对大块自由空间合并可能造成的妨碍，可以针对大小不一的对象提供多个堆；优先将相关的分配动作和归还动作保持在一起，这使得分配所得好像是连续的，而归还动作会释放所有连续空间，为下一次分配建立一个大块连续空间。[2]

3.3.2 分区管理

内存分区是应对碎片问题和实现动态内存管理的一个常规方法。

固定分区法在系统启动时把主存简单地划分为 n 个不一定相等的分区，设置好后就不再被改变，把到达进程放到能够容纳它的最小的可用分区输入队列中。固定分区易于理解也易于实现，只需很小的开销。其不足是分区和分块数目在系统生成时确定，限制了活跃进程的数目，灵活性不够，且如果可用分区的大小与进程大小非常不匹配，会产生比较大的内部碎片，空间的利用率很低。

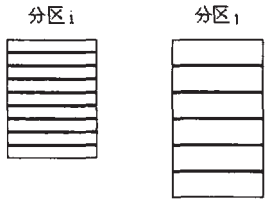

图 2 改进型分区管理中的分区

动态分区利用率随着时间的推移、越来越多空洞和碎片的出现而下降，且维护十分复杂，需要非常费时的存储器压缩以克服外部碎片开销，一般不使用。

μC/OS-II 使用的就是如图 2 的改进型分区管理。它将连续的大块系统内存分为多个区，各区含有整数个大小相同的内存块，不同内存区的内存块大小不同，应用程序可根据需要选取不同的内存块或内存块组合。内存释放时，特定内存块必须释放回原来所属内存区。利用这种机制，μC/OS-II 对 malloc()和 free()函数进行了改进，使得它们可以在固定执行时间内分配和释放固定大小的内存块，并有效避免了内存碎片的产生。[4]

3.3.3 池式分配

内存池很象分区，应用于使用大量相似对象、并经常进行分配和归还动作的场合。池式分配法预先分配足够内存，通过静态分配的数组建立起内存池。一旦需要一个新对象，便在池中选择一个尚未被使用的对象，由程序初始化并使用之。不再需要时，便还入内存池。池式分配用一个内嵌指针的链表跟踪池中对象状态，比系统堆中的分配、归还要方便、快捷得多。

3.3.4 简单分页

简单分页类似固定分区，主存分成许多大小相等且很小的帧，每个进程分成同样大小的页，当一个进程被装入时，它的所有页被装入到可用的但并不需连续的帧中，并建立一个页表。进程每一页在页表中有一个对应项，每个页表项包含主存中保存相应页的帧的帧号，页表按页号进行索引。此外，一个自由帧列表保存着当前主存中未被占用、可供页使用的所有帧。

μClinux 就采取了简单分页的实存储器管理策略。

3.4 交换和虚拟存储

一般来说，多数嵌入式系统内存可容纳所有进程，一旦一个任务被装入内存，它将一直保持在原来的位置直到运行结束，它永远不会被换出或移动到内存其他位置，为它分配的空间也不会增长或缩小。交换不是必要的，它会增加系统的复杂性。

加载缺页会严重影响对于实时系统很关键的响应时间和可预测性，必须考虑到虚拟内存对实时性的影响。而且，大多数嵌入式系统使用无存储管理单元（MMU）的处理器，直接访问内存物理地址。对兼容性的考虑要求内存管理方案尽量简单。如果使用了虚拟等技术，则要移植到没有这些特性的非高端处理器上将十分困难。通过对硬件所能提供的支持作最少的假设可以使其能移植到占绝大多数的中低端平台上。

3.5 可配置内存管理技术

没有什么绝对的最佳解决方案，只存在特定情况下更为合适的策略，内存管理也是这样。

大多数嵌入式操作系统只支持单一的内存管理策略，无法根

据实际情况做出相应调整，这给其移植、应用带来了不小的问题。RTEMS 等操作系统虽然同时提供了多种内存管理方式，但其复杂的数据结构既占用了较多内存，也给开发带来了一定困难。

采用可配置、易裁剪的嵌入式操作系统内存管理方式，基本可解决上述问题。操作系统设计者提供多种典型情况下较好的内存管理模块，使用者根据应用环境，通过设置条件编译选项、配置文件等方式从中选择最合适的加以应用。一般提供全静态、μC/OS-Ⅱ式分区、虚拟等两三种就已足够。为修改方便考虑，这样的内存管理模块往往不是内核的一部分，而是作为运行在用户空间、使用标准的消息机构和内核通信的外部服务进程来实现的。

4 结束语

操作系统设计者和开发使用者应该根据实际情况通盘考虑、权衡利弊，明智地选取内存管理策略以解决问题。若存在多个满足需求的可用解决方案，那么一般从中选择最简单的那个，以避免引入不必要的复杂性。

参考文献：

[1]Tanenbaum A.Modern operating systems [M].Beijing:China Machine Press,2002.190-262.

[2]Noble J,Weir C.侯捷,等,译.内存受限系统之软件开发:针对内存受限系统而整理的模式 [M]. 武汉：华中科技大学出版社,2003.219-259.

[3]Stallings S.魏迎梅,等,译.操作系统:内核与设计原理[M].北京:电子工业出版社,2001.225-289.

[4]Labrosse J.邵贝贝,等,译.嵌入式实时操作系统 μC/OS-Ⅱ[M].北京:北京航空航天大学出版社,2003.270-283.

(上接第 139 页)

```
    ws1.Cells(site_index, 4).Value = site_row
    ' 在表 1 第 4 列显示该行号
    End If
    ' 以下为将表 2 中的基站信息数据写到表 3 相应的单元格
    For i = 1 To 3
    ws3.Cells(site_hang, 2).Value = ws2.Cells(site_row, 2).Value
    ws3.Cells(site_hang, 3).Value = ws2.Cells(site_row, 4).Value
    ws3.Cells(site_hang, 4).Value = ws2.Cells(site_row, 6).Value
    ws3.Cells(site_hang, 5).Value = ws2.Cells(site_row, 7).Value
    ws3.Cells(site_hang, 6).Value = ws2.Cells(site_row, 8).Value
    site_hang = site_hang + 1
    site_row = site_row + 1
    Next i
    ws3.Cells(site_hang - 3, 1).Value = ws1.Cells(site_index, 1).Value
    site_index = site_index + 1    ' 表 1 中的下一个基站
    Loop
    End Sub
```

以上代码的工作原理是以表 1“基站名称(转化之后)”列中的基站名称为检索字，在表 2 的第 4 列“基站名称”中查找，如果找到该基站，则将表 2 中该基站的相关数据写到表 3 中相应的单元格中，并把该基站在表 2 中的行号写到表 1 的第 4 列。本代码的关键技术在于 Range 对象 find 方法的使用。Find 方法用于在区域内查找特定信息，并返回 Range 对象，该对象代表所找到的第一个包含所查找信息的单元格。如果未发现匹配单元格，就返回 Nothing。比如要在工作表 1 的单元格区域“A1:A500”中查找包含值 2 的单元格，并将这些单元格变灰，代码如下：

```
    With Worksheets(1).Range("a1:a500")
    Set c = .Find(2, lookin:=xlValues)
    If Not c Is Nothing Then
    firstAddress = c.Address
    Do
    c.Interior.Pattern = xlPatternGray50
    Set c = .FindNext(c)
    Loop While Not c Is Nothing And c.Address <> firstAddress
    End If
    End With
```

到此，已经完成了“生成”代码的编写，可以为“生成”按钮指定宏 makesheet 了，指定宏之后，单击“生成”按钮，30 个无线基站的筛选结果数据在短短 1 秒钟之内就处理完成，如图 4 和图 5 所示是“主程序”工作表的前 9 行和生成的“筛选结果”工作表的前 7 行。

	A	B	C	D	E	F
1	序号	基站名称(转化之后)	基站名称(拷贝)	该基站在“无线基站信息”表中的行号		
2	1	安良d	安良(d)	2792	转化	
3	2	工会d	工会(d)	188		
4	3	建华d	建华(d)	2807	生成	
5	4	茶山m	茶山(m)	2792		
6	5	红岭d	红岭(d)	3257	使用说明：	
7	6	天健d	天健(d)	3269	1.将基站名称列拷贝过来	
8	7	安托山m	安托山(m)	2162	2.转化	
9	8	民乐村d	民乐村(d)	1544	3.生成	

图 4 “主程序”工作表前 9 行

以上完成了工作表界面的设计还有所有宏程序代码的编写。以往要从“无线基站信息”工作表中筛选出 30 个基站的相关信息，往往需要半个小时的时间。现在利用该程序，即使要筛选 200 个无线基站的信息数据，只需从“无线基站清单”工作表中把基站名称拷贝过来，“转化”然后“生成”，则 200 个基站的数据在 1 秒内就自动处理完成了。

	A	B	C	D	E	F
1	序号	合同编号	基站名称	天线方向	AM9载波数	基站类型
2	1	SZ3OD1001	安良d	70°	4	RBS2202
3				150°	4	
4				340°	4	
5	2	SZ3OO2015	工会d	65°	10	RBS2202
6				160°	12	
7				290°	3	

图 5 “筛选结果”工作表前 7 行

3 结论

本文介绍了 VBA 在处理无线基站信息数据方面的一个具体应用，实践证明极大地提高了数据处理的效率。通过对宏代码改写和补充，还可将本方法应用于其他需要在大量数据中查找筛选信息的领域，大大地加快数据处理的速度和减少错误的发生，提高工作效率。VBA 具有简单易学、句法简单的特点，而且不需要额外安装 VB 的开发环境，因此，无论在实际工程领域，还是日常工作的数据处理方面，都有非常广阔的应用前景。

参考文献：

[1]晶辰工作室.Excel 2000 VBA 开发实例指南[M].北京:电子工业出版社,2000.

[2]陈向荣.VB 与 Excel 二次开发[J].福建电脑,2005,(3).

[3]Microsoft Visual Basic 帮助文档[Z].