嵌入式操作系统内存管理机制分析及改进

孙益辉 陈 凯 白英彩
(上海交通大学信息安全工程学院 上海 200030)

摘 要 本文介绍了嵌入式操作系统内存管理基本原理,分析了 μClinux内存管理机制实现,针对其内存管理不足,提出了改进和实现的方法。

关键词 嵌入式操作系统 内存管理 μClinux

STUDY AND IMPROVEMENT OF EMBEDDED OPERATING SYSTEM MEMORY MANAGEMENT MECHANISM

Sun Yihui Chen Kai Bai Yingcai
(*School of Information Security Engineering, Shanghai Jiaotong University, Shanghai 200030, China*)

Abstract This article introduces the basic theory of memory management in embedded operating system and analyses the realization of μClinux. It also presents a method of improvement and realization of memory management as for its defaults.

Keywords Embedded operating system Memory management μClinux

1 引 言

嵌入式系统是以应用为中心,计算机技术为基础,软硬件可裁剪,适应于特定应用系统,对功能、可靠性、成本、体积、功耗有严格要求的计算机系统。嵌入式系统应用相当广泛,从日常生活中的电子产品如数码相机、DVD播放器到航空航天器的导弹制导系统、飞行控制系统等,都可以看到嵌入式系统的影子。

常用嵌入式操作系统按实时性能可以分为两类。一类是面向控制、通信领域的强实时操作系统,如 WindRiver公司的 VxWorks pSOS, QNX系统软件公司的 QNX, AT的 Nucleus等;另一类是面向消费类电子产品的弱实时操作系统,它包括个人数字助理(PDA)、移动电话、机顶盒、电子书、WebPhone等,应用比较多的有 Windows CE, Palm OS。此外,还有一些源代码公开的嵌入式操作系统,如 μC/OS, μClinux等,因为其代码公开,便于实现移植和扩展,应用也很广泛。

2 嵌入式操作系统内存管理特征

嵌入式系统是一个资源受限系统,系统对内存开销和管理比其它系统更严格。由于受资源、成本等限制,嵌入式系统没有庞大的内存管理单元,程序员在设计程序时需考虑内存分配和使用问题,嵌入式系统内存管理从整体上来说有以下几个特征。

快速性 嵌入式操作系统实时性强,要求内存管理机制反应快,因此在嵌入式系统中,一般不采用通用操作系统中复杂而完善的内存管理策略,而是采用简单、快速的内存分配和管理方案。当然,对实时性要求的程度不同,内存管理机制方案也有所不同。

可靠性 它是嵌入式系统内存管理机制的基本要求,如果内存管理失败,可能会带来严重后果,特别是应用于一些关键设备或部件上的嵌入式系统,如汽车、航天器等。

高效性 内存管理机制要尽可能地少浪费内存资源,不能为了满足所有内存请求而将内存配置很大。嵌入式系统要求对成本进行限制,其次是嵌入式系统有限的空间也决定了内存容量配置是有限的。

3 嵌入式操作系统内存管理分析及改进

嵌入式系统一般没有存储管理部件 MMU 硬件支持单元,不能进行虚拟地址转换,地址总线上的地址全部是实地址,内存管理机制相对简单,下面就嵌入式操作系统 μClinux内存管理机制进行具体分析。

μClinux仍然沿用 linux按页管理方式,将整个物理内存划分成大小为 4KB的页面。由数据结构 page管理,有多少个页面就有多少个 page结构,它们作为元素组成一个数组 mem_map[]。

内存模块初始化主要包括三个函数:paging_init()、free_area_init()和 mem_init()。

paging_init()在标准 linux中需要做很多操作,但是 μClinux中它只为系统启动时保留一部分特殊用途的内存区间,paging_init()函数在返回前通过调用 free_area_init()建立 buddy system映射位图表以及建立空闲物理页面链表。

paging_init()函数主要内容如下:

```
empty_bad_page=(unsigned long*)start_mem;
start_mem+=PAGE_SIZE;
empty_zero_page=(unsigned long*)start_mem;
```

收稿日期:2004-08-06。孙益辉,硕士生,主研领域:嵌入式系统。

```
start_mem+=PAGE_SIZE;
memset((void*)empty_zero_page, 0, PAGE_SIZE);
return PAGE_ALIGN(free_area_init(start_mem, end_mem));
```

函数执行完后,内存中依次是:empty_bad_table、empty_zero_page以及 mem_map[ ]bitmap。

free_area_init( )函数用于建立管理物理页面的数据结构 mem_map,有多少物理页面就有多少 mem_map_t类型的结构体与之相对应。每个页面的 mem_map_t结构中的 flag被标明为 PG_DMA 和 PG_reserved,并且页帧号被赋给相应的数值,同时建立管理页面的 bitmap映射表,所有位被置为"0"。

mem_init( )函数遍历整个可用物理内存地址空间,统计可用物理页面数量,打印系统各个内存参数,如内核代码段和数据段大小等,当 mem_init( )遍历整个可用物理内存地址空间时,把每个空闲页面 count置为"1"。然后再调用 free_page( ),把空闲页插入到相应 free_area成员链表中。

空闲页面由 get_free_pages( )函数和 free_pages_ok( )函数分配和回收,采用 Buddy(伙伴)算法。Buddy算法的原理是每次分配包含一个或者多个物理页面的内存块,页面是以 2的幂次大小内存块来分配。首先搜寻满足请求大小的页面,它从满足当前申请大小的 Buddy数据结构域搜索空闲页面,如果没有请求大小空闲页面,则它搜索两倍于请求大小的内存块。这个过程一直持续到所有 Buddy被搜索完或找到满足要求的内存块,如果找到的页面块大于请求块,则对其进行分割以使其大小与请求块匹配。释放时,先检查该块的伙伴使用情况,如果没有被使用,则合并这对 Buddy,再检查该两倍大小块的 Buddy 一直持续到合并后的块没有 Buddy为止,并将它链到该大小的 freepage队列中。

通过 get_free_pages( )只能分配最小为 4KB的页块,每次申请内存时未必需要这么大,为减少内存浪费,μClinux提供了 kmalloc( )和 kfree( )两个函数。它们是建立在空闲内存块之上的一个算法,能提供最小 1字节的空间。进程可以向核心申请使用物理内存,这些内存块来自于 free_area数组,由 blocksize表、sizes表、page_descriptor和 blockheader结构共同管理。

block_header用来管理单个数据块,数据块大小有很多种,由常量数组 blocksize[ ]决定,block_header的结构如下:

```
struct block_header
  {
  unsigned long bh_flags;
  union{
     unsigned long ubh_length;
     struct block_header* fbh_next;
  } vp;
};
```

结构成员 union vp分别代表两种功能,当此 block空闲时用 fbh_next成员,使其链入到相应的链表中;当此 block被分配后,用 ubh_length成员表示 block大小。

get_free_pages( )申请的是页面,申请到页面后会把它分成小的 block,block用 block_beader结构来管理,而页面则用 page_descriptor管理,page_descriptor的结构如下:

```
struct page_descriptor{
     struct page_descriptor next;
     struct block_header firstfree;
     int order;
     int nfree;
};
```

next成员用来把结构变量链接到相应链表中,firstfree则指向这个页面第一个空闲块,order是指包含 block大小的权,nfree用来是指示当前页面的空闲 block数量。

kmalloc( )执行过程:当没有该大小的空闲 block时,kmalloc从 free_area中申请页面,申请到后,在页面开始位置填上 page_descriptor结构管理页面,然后把剩下的单元组织成 block_header结构,再把页面链接到 size[ ]数组的相应成员中。sizes[ ]数组的成员为 size_descriptor结构类型,当页面的 block被分配完以后,再把该页面从相应的 sizes[ ]数组成员的链表删除,size_descriptor的结构如下:

```
struct size_descriptor
{
  struct page_descriptor firstfree;  /*指向第一页*/
  struct page_descriptor* dmafree;
  int nblocks;  /*该组每页面含 block的数量*/
  int nmallocs;  /*该组已分配 block数量*/
  int nfrees;  /*该组回收 block的数量*/
  int nbytesmalloced;  /*实际分配字节数*/
  int npages;  /*该组所包含的面框数*/
  unsigned long gfporder;  /*该组 block所占的页面数(block不足一页按一页算)的权(order)*/
};
```

它们之间的关系如图 1所示。

kfree( )执行过程:根据释放地址找到页面 page_descriptor指针 page和 block_header指针,然后根据 page.order找到该页对应的 sizes[ ]组。把该 block标记为空闲,然后把它链接到该页空闲 block链表中。更新 sizes[order].nfree以及 sizes[order].nbytesmalloced;如果该页在之前没有空闲 block,则把该页链接到 size[order]中,如果该页在此时所包含的 block全部空闲,则释放该页。

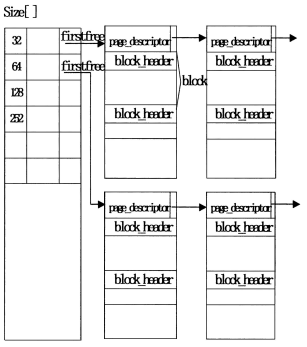


图 1　μClinux数据结构之间关系

在 μClinux中,用户可以通过 kmalloc( )和 kfree( )来申请和释放小块内存,但在实现中,它通过数组 size[ ]决定每次申请内存的大小,申请的最小单元实际是 32字节,在应用中,并不是每次需要这么大的空间,浪费了嵌入式系统中宝贵内存资源;其次是在申请过程中,程序采用了从小到大依次比较的方法,执行效率并不高。

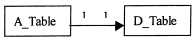
针对 μClinux嵌入式操作系统中问题,笔者在某个实际项目中对它进行了改进,以达到减少浪费和提高效率的目的。首先是修改 blocksize[ ]数组和 size[ ]数组,使申请的实际最小内存单元为 16字节或者更小,这只需在 blocksize[ ]数组和 size[ ]数组中增加元素,使每个页面容纳更多 block。

其次是在 size[ ]数组中,所分配空间的大小已经是有序的,可以采取优化算法来提高查找效率,如折半查找方法,具体实现如下:

```
while(min<=max)
{
  mid=(min+max)/2
```

(下转第 115页)

```
……
<xsd: element name="D_Table">
    <xsd: complexType>
    <xsd: sequence>
    ……
</xsd: sequence>
</xsd: complexType>
```



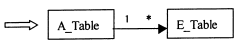
映射为：

```
TABLE A_Table;
CONSTRAINT A_PKEY id PRIMARY KEY
TABLE D_Table;
CONSTRAINT D_PKEY id PRIMARY KEY
CONSTRAINT D_A_REF FOREIGN KEY A id REFERENCES
A_Table( id) ON DELETE CASCADE
CONSTRAINT UN_AID A id UNIQUE
```

④ 复杂类型 A包含多个拥有复杂类型 E的元素

拥有复杂类型 E的元素包含在复杂类型 A中，拥有类型 A的其它元素不能共享这些类型为 E的子元素，因此这是一个一对多的关系，约束关系为当类型为 A的元素被删除时，包含于其中的类型为 E的子元素也一起被删除。例如下面 XML模式：

```
<xsd: complexType name="A_Table">
  <xsd: sequence>
    ……
    <xsd: element name="E_Table">
      <xsd: complexType>
        <xsd: sequence>
        ……
</xsd: sequence>
<xsd: element name="E_Table">
      <xsd: complexType>
        <xsd: sequence>
        ……
</xsd: sequence>
</xsd: complexType>
```



映射为：

```
TABLE A_Table;
CONSTRAINT A_PKEY id PRIMARY KEY
TABLE E_Table;
CONSTRAINT E_PKEY id PRIMARY KEY
CONSTRAINT E_A_REF FOREIGN KEY A id REFERENCES
A_Table( id) ON DELETE CASCADE
```

4 结束语

对 XML模式与关系模式映射算法的研究目前主要集中在国外，如加利福尼亚大学的 Dongwon教授在采用基于 XML的关系数据库发布和基于关系数据库的 XML存储和检索技术的基础上，提出并初步建立基于约束的模式映射方法；但是很多算法的研究目前也只停留在基于 DTD模式而非 Schema模式，这是一个需要进一步研究的方向。由于 Schema取代 DTD只是时间上的问题，因此，基于 Schema的 XML模式与关系模式映射算法的研究将会成为今后进一步研究的重点。

参 考 文 献

[1] Irakl is Varlam is M ichalis Vazirgiannis Bridging XML-Schema and relational databases, a system for generating and manipulating relational databases using valid XML documents Proceeding of the ACM Symposium on Document Engineering Atlanta Georgia USA, 2001, 105～110

[2] Igor Tatarinov, Stratis D. Viglas Storing and Querying Ordered XML Using a Relational Database System, Proceedings of the 2002 ACM SIGMOD international conference on Management of data Madison Wisconsin USA, 2002, 204～214

[3] S. Netorov, S. Abiteboul, R. Motwani, Extracting Schema for Semistructured Data Proc of ACM SIGMOD Conference on Management of Data Seattle, WA, 1998.

(上接第 99页)

```
if GT( realsize, BLOCKSIZE(m id－1))
        &&LT( realsize, BLOCKSIZE(m id))
    return m id;
else if LT( realsize, BLOCKSIZE(m id))
    m ax=m id－1;
else m in=m id+1;
}
```

μC linux中，进程维护本身内存地址空间方法与标准 linux不同，它用一种更简单实用的方法来实现，即在 mm_ struct中维护一个内存块链表，通过两个结构 struct mm_ rblock+ struct和 struct mm_ rblock_ struct实现，mm_ rblock_ struct把 mm_ rblock_ struct结构所描述对象组成一个链表，mm_ rblock_ struct则用来管理内存块，它的具体结构如下：

```
struct mm_ rblock_ struct
{
    int size;
    int refcount;
    void *  kblock;
};
```

size说明 kblock所指向内存区域的大小，refcount记录此内存空间用户个数。kblock是指向这个内存块空间起始位置指针。

4 结束语

嵌入式系统一般没有 MMU硬件支持，不支持虚拟内存管理和内存保护，没有页表和页目录对线性地址的映射，从线性地址到物理地址转换无需做任何工作，因此，它的响应速度快。此外，由于没有虚拟内存管理功能，μC linux不再使用“按需调页”，程序在载入内存执行时需将程序的全部映像一次装入，比物理内存大的程序将无法执行。

参 考 文 献

[1] 李善平、刘文峰、王焕龙，Linux与嵌入式系统[M]，北京：清华大学出版社，2003.

[2] James Noble, Charles Weir, Duane Bibby. Small Memory Software: Patterns For System With Limited Memory[M]. Boston: Addison-Wesley Professional, 2000.

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

[4] 卢春鹏，“一种嵌入式系统的内存分配方案[J]”，《单片机与嵌入式系统应用》，2002, 12

[5] 崔文杰，嵌入式操作系 μC linux[EB/OL]，http://www.51eda.com/Article/embed_system/rtos/200401/543.htm, 2004－01－03.