静态内存管理系统的研究与应用

戚海燕

(浙江经济职业技术学院经济信息系，浙江 杭州 310018)

摘 要：静态内存管理系统与动态内存管理系统是嵌入式操作系统中常见的两种内存管理方式。文章着重分析了静态内存管理系统的原理、数据结构和应用函数，然后以数字电视广播系统多路接收器为例介绍了其在嵌入式系统中的应用，最后就它的几个优点进行了阐述。

关键词：静态内存管理；嵌入式系统；内存链表；内存结点

0 引言

静态内存管理系统是一种管理内存的算法，它把一整块内存区域分成等大小的几个内存块，然后以空内存链表和满内存链表来管理每一个内存块。除了正在被系统模块使用外，内存模块只可能在空内存链表或者满内存链表中。这样能够充分地利用每一个内存块。但是因为每块内存的大小必须一样，导致它的使用不够灵活。在嵌入式系统中很多模块的输入输出缓冲区可以被分成等大小的内存块，这样我们就可以用静态内存管理系统很方便地去管理嵌入式系统各模块的输入输出内存区。

1 静态内存管理系统的原理概述

静态内存管理系统在整个系统初始化时根据各系统模块的特性决定各模块所需的内存池大小，并把它分成若干个相同大小的内存块，再以链表的形式把它们串成一个内存链来管理。当内存管理系统收到内存使用需求申请时，就会从内存链中分配出一个内存块供相应模块使用。当该内存块被使用完毕，它就被释放回其所属的内存链。

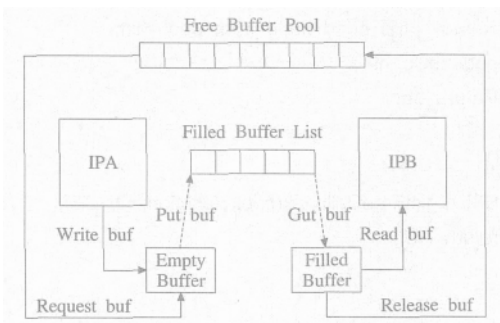

图 1 静态内存管理系统整体架构

如图 1 所示，系统模块 IPA 和系统模块 IPB 是读写相关的两个模块，IPA 的输出缓存是 IPB 的输入缓存。当系统初始化时我们把 IPA 的输出缓存即 IPB 的输入缓存看作是一个空内存池，并把它分成相同大小的几块内存区域。每块内存区域由一个内存结点来管理，然后把各结点连接成一个链表。当 IPA 需要输出数据时，即从这个空内存池链表中申请一个内存结点，并把所输出数据写到该内存结点对应的内存区域中，当该内存结点的区域被写满时，IPA 就把此内存结点添加到另一个内存链表中，我们称此内存链表为满内存链表。此满内存链表所管理的内存区域就是 IPB 的输入缓存。当 IPB 需要读数据时，就从此满内存链表中得到一个内存结点，并读取此结点所对应内存中的数据。当 IPB 读完此结点所对应内存中的数据后，即把此结点释放回上面提到的空内存池链表中。

2 静态内存管理系统中的数据结构

(1) 内存结点即内存控制块是静态内存管理系统中最基本的单元，我们用它来管理一定大小的一段内存空间。如图 2 所示。

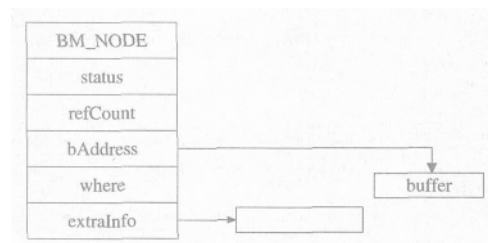

图 2 内存控制块示意图

status: 是该结点所对应内存区域的状态，分为 FREE/ALLOC 两种状态。FREE 状态是指该内存结点没有被任何系统模块所使用，即该内存结点在空内存池链表中；ALLOC 状态是指该内存结点没在其所属的空内存池链表中，而是被系统模块读写数据或者在满内存链表中。

refCount: 是该内存结点被引用的次数。当该结点在空内存池链表中时，其 refCount 值为 0；当该结点在空内存池链表中刚被申请出来时，其 refCount 值为 1；当该结点被放入某个满内存链表中时，其 refCount 被加 1；当该结点被系统模块释放时，其 refCount 被减 1；当该结点的 refCount 被减为 0 时，此结点被释放回其所属的空内存池链表。

bAddress: 是该内存结点所对应的内存区域的地址指针。

where: 是该内存结点所属的空内存池链表的指针。

extraInfo: 是一个 void 型的指针。它可以指向任意类型的区域，而该区域可以包含此内存结点应有的一些特殊的信息。

(2) 空内存池链表用来管理多块空内存的数据结构，如图 3 所示。当系统模块要输出数据时它需从空内存池链表中申请(request)一个内存结点；当系统模块把一个内存结点的数据读空后，就把此内存结点释放(release)到空内存池链表中。

valid: 表示该空内存池链表的有效性，分为 INVALID/VALID 两种状态。INVALID 说明该空内存池链表还未被初始化不能使用；VALID 说明该空内存池链表已被初始化可以使用。

bSize: 表示该空内存池链表中每个内存结点所对应的内存区域的大小。

bTotalCount: 表示该空内存池链表初始化时，所含内存结点的总数。

bCount: 表示该空内存池链表当前所含有的内存结点的数量。

start: 表示该空内存池链表所代表一整块内存池的起始地址。

function: 是一个函数指针，指向系统模块把内存结点释放回空内存池链表时所需调用的回调函数。

callbackType: 表示此空内存池链表的回调函数的调用方式，分为 EVERYTIME/ONCE/NULL 三种方式。EVERYTIME 表示此回调函数在每次内存节点被释放回此空内存池链表时都要被调用一次；ONCE 表示此回调函数只在空内存池链表为空后收到第一个被释放回的内存结点时调用一次；NULL 表示此回调函数不会被调用。

list: 是一个链表结构指针。该链表结构包含头内存结点指针和尾内存结点指针。此链表结构指针指向空内存池链表所代表的一串内存结点。

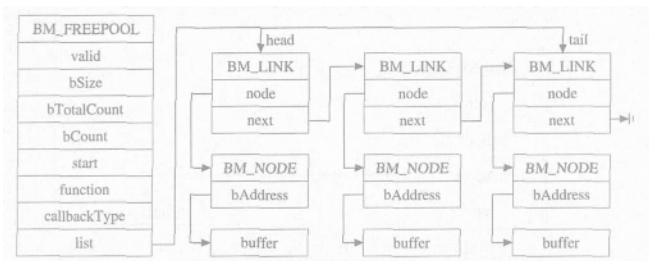

图 3 空内存池链表结构

(3) 满内存链表用来管理多块充满有用数据内存的数据结构。如图 4 所示，当系统模块写满一个内存结点后，就把它放入(put)满内存链表中。当系统模块要输入数据时它需要从满内存链表中得到(get)一个内存结点。其各项元素的含义同空内存池链表的相似。

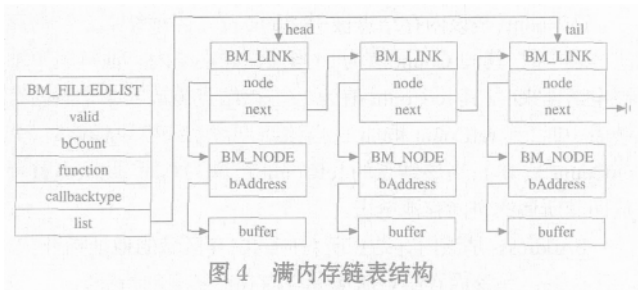

图 4 满内存链表结构

3 静态内存管理系统中的应用函数

3.1 空内存池链表与满内存链表的回调函数注册

当需要把特定的函数指定为空内存池链表和满内存链表的回调函数时，我们通过函数 BM_registFreeCallback 注册空内存池链表的回调函数，通过 BM_registFilledCallback 注册满内存链表的回调函数。

下面是 BM_registFreeCallback 和 BM_registFilledCallback 的函数定义：

```
VOID BM_registFreeCallback (BM_FREEPOOL *pool,
  BM_FILLEDCALLBACK function, BM_CallbackType_et type)
{ pool->function = function;
  pool->callbackType = type;
}
VOID BM_registFilledCallback (BM_FILLEDLIST *list,
  BM_FILLEDCALLBACK function, BM_CallbackType_et type)
{ list->function = function;
  list->type = type;
}
```

3.2 空内存结点的申请与释放

当系统模块需要输出数据到内存区域时，通过调用 BM_requestFreeBuffer 向相应的空内存池链表申请内存结点。如果空内存池链表不为空则返回链表的头结点指针给系统模块并从空内存池链表中删除该结点；反之返回空指针，同时根据需要给空内存池链表注册相应的回调函数，当空内存池链表收到释放回的结点时调用此回调函数。

系统模块读空内存结点后，需要通过调用 BM_releaseFreeBuffer 释放此内存结点。如果该内存结点的 refCount 在减 1 后仍大于 0，说明此内存结点还在被其他的系统模块使用，暂时不能释放回它相应的空内存池链表；如果该内存结点的 refCount 在减 1 后等于 0，说明没有系统模块使用此内存结点，它就被加入到空内存池链表的尾部，即被释放回其所属的空内存池链表，同时根据空内存池链表的回调函数是否为空和回调类型是 EVERYTIME 还是 ONCE 决定是否调用相应的回调函数。

```
BM_NODE* BM_requestFreeBuffer(BM_FREEPOOL* pool,
  BM_FREECALLBACK function)
{ if (pool->bCount > 0)
  { tmp = pool->list->head;
    tmp->refCount = 1;
    Remove the head node from pool->list;
    pool->bCount--;
    Return tmp;
  }
  else
  { BM_registFreeCallback(pool, function, ONCE);
    return NULL;
  }
}
VOID BM_releaseFreeBuffer(BM_NODE*node)
{ node->refCount--;
  if (node->refCount != 0) return;
  put node to the tail of node->where->list;
  node->where->bCount++;
  if(node->where->function != NULL)
  { if(node->callbackType == ONCE)
    { if (node->where->bCount ==1)
      { Call node->where->function;
        node->where->function = NULL;
      }
    }
```

```
        else if (node->callbackType == EVERYTIME)
        { Call node->where->function;
        }
    }
}
```

3.3 满内存结点的放入与获得

当系统模块需要输入数据时，通过调用 BM_getFilledBuffer 从它自己相对应的满内存链表中得到满内存结点。如果满内存链表不为空，则返回链表的头结点指针给系统模块并从满内存链表中删除该结点；反之返回空指针，同时根据需要给满内存链表注册相应的回调函数，当满内存链表收到放入的结点时调用此回调函数。

当系统模块写满一个内存结点对应的内存区域后，根据需要把结点放入相应的满内存链表的尾部，同时根据满内存链表的回调函数是否为空和回调类型是 EVERYTIME 还是 ONCE 决定是否调用相应的回调函数。此过程是通过调用 BM_putFilledBuffer 来实现的。

```
BM_NODE* BM_getFilledBuffer(BM_FILLEDLIST*list,
BM_FILLEDCALLBACK function)
{ if (list->bCount > 0)
  { tmp = list->list->head;
    Remove the head node from list->list;
    list->bCount--;
    return tmp;
  }
  else
  { BM_registFilledCallback(list, function, ONCE);
    return NULL;
  }
}
VOID BM_putFilledBuffer(BM_FILLEDLIST*List,BM_NODE*node)
{ node->refCount++;
  put node to the tail of List->list;
  list->bCount++;
  if (list->function != NULL)
  { if (list->callbackType == ONCE)
    { if (list->bCount == 1)
      { call list->function;
        list->function = NULL;
      }
    }
    else if (list->callbackType == EVERYTIME)
    { call list->function; }
  }
}
```

4 静态内存管理系统在嵌入式系统中的应用

图 5 是一个应用静态内存管理系统的嵌入式系统的实例——数字电视广播系统的多路接受器系统架构图。总线接收端口从信道上接收恒定大小（188 字节）的数据包，并把它按包头里的信息传送给相应的一个或多个用户端口。总线接收端口的空内存池链表被分成 188 字节大小的若干个结点，当总线接收端口接收到一个数据包后，它首先通过调用 BM_requestFreeBuffer 从其空内存池链表中申请一个内存结点，同时为了处理空内存池链表为空的情况，把写数据包的函数注册成空内存池链表的回调函数，然后调用写数据包函数把数据包写入此内存结点所对应的内存区域。如果接收到的数据包需要发送给端口 1、端口 2 和端口 3，则通过调用 BM_putFilledBuffer 将此内存结点分别放入端口 1、端口 2 和端口 3 所对应的满内存链表。此时该内存结点的 refCount 已经变成 4，为了能够把此结点最终释放回空内存池链表中，应提前调用 BM_releaseFreeBuffer 一次使该结点的 refCount 变为 3。当用户端口 n（n= 1，2，3）需要接收数据包时，首先调用 BM_getFilledBuffer 从其满内存链表中获得内存结点，同时为了处理满内存链表为空的情况，把读数据包的函数注册成满内存链表的回调函数，然后调用读数据包函数从内存结点中读取数据包，在读完之后调用 BM_releaseFreeBuffer 释放此内存结点。当用户端口 1、端口 2 和端口 3 都调用了 BM_releaseFreeBuffer 释放了同一个内存结点后，此结点即被释放回空内存池链表中。可以看出，通过使用静态内存管理系统，本来读写相关的总线接收端口和用户端口 n（n= 1，2，3）被完全隔离开，简化了它们对内存的使用方法。

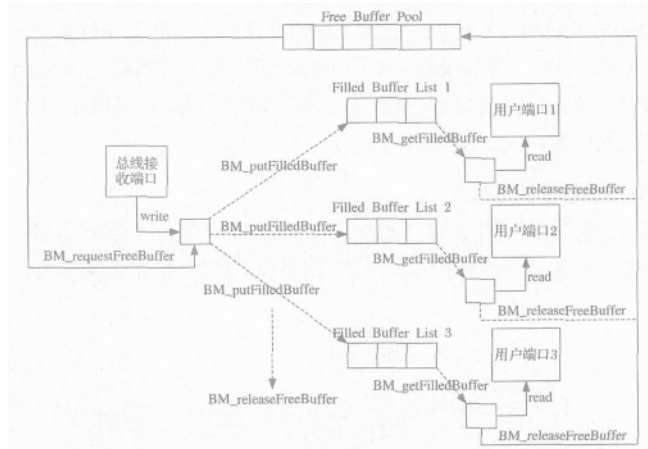

图 5 数字电视广播系统多路接收器

5 结束语

在介绍了静态内存管理系统的原理、数据结构、应用函数和应用实例后，可以看出静态内存管理系统的几个优点：一、效率高，仅用两个链表结构就可以管理所有空内存和满内存；二、容易处理特殊情况，在处理一块内存被多个模块所引用时，可以无需考虑各模块间的关联性；三、模块化，读内存模块和写内存模块都只需和静态内存管理系统打交道；四、节省内存资源，通过回调函数的方法，在空内存池链表或满内存链表空后，可以第一时间处理被释放回或放入的第一个内存结点。

参考文献：

[1] 田令平.嵌入式操作系统内存管理研究.电脑知识与技术（学术交流），2006.4:161

[2] 孙益辉，陈凯，白英彩.嵌入式操作系统内存管理分析及改进.计算机应用与软件，2006.23(3):98