文章编号：2096-1472(2016)-04-59-02

一种微型嵌入式系统动态内存分区管理机制的研究

肖 蕾，刘克江
（广东技术师范学院自动化学院，广东 广州 510635）

摘 要：本文基于现有嵌入式操作系统动态内存管理机制的原理，研究和分析了μC/OS、freeRTOS等多个微型嵌入式操作系统内存管理的优缺点，以多平台通用、高效率、简单易用为目标，实现了一种新的动态内存管理机制，主要适用于不使用操作系统或使用小型操作系统的嵌入式产品中，有效解决了动态内存的管理问题。

关键词：嵌入式系统；动态内存；内存溢出；管理机制
中图分类号：TP316.2 **文献标识码**：A

Research on a Dynamic Memory Management Mechanism of Micro Embedded System

XIAO Lei,LIU Kejiang
(*College of Automation Engineering,GuangDong Polytechnic Normal University,Guangzhou* 510635,*China*)

Abstract: Based on the principle of existing embedded operating system dynamic memory management mechanism, research and analysis of the advantages and disadvantages of μC/OS,freeRTOS and other micro embedded operating system memory management.In this paper,multi-platform,highly efficient,easy to use as the goal to achieve a new dynamic memory management mechanism, mainly applicable to not use the operating system or the use of small embedded operating system products,the mechanism effectively solves the problem of dynamic memory management.

Keywords:embedded system;dynamic memory;memory overflow;management mechanism

1 引言(Introduction)

嵌入式操作系统内核往往提供了动态内存分区管理机制，基于操作系统移植便可以在应用开发时使用动态内存分配，使用动态内存分配可避免使用大量内存空间，从而使软件设计更加合理。但是有时由于成本、硬件设计或功能需求等因素不需要移植嵌入式操作系统，或者只需使用μC/OS、freeRTOS等微型操作系统内核，就会发现动态内存分区管理机制存在着许多不足[1]。μC/OS为了尽可能精简代码和追求更好的实时性，其内存管理模块根据用户程序要求把动态内存分成多个分区，每一个分区又分成数量和大小固定的内存块，内存块数量和大小在编程时便已确定不可更改，限制了系统的扩展性和灵活性，而且由于μC/OS内存管理模块创建一个动态内存分区只能提供唯一尺寸的内存卡，当系统需要不同尺寸时须创建多个动态内存分区，这样便不可避免地加大了系统开销和内存浪费，且μC/OS不对回收的内存块进行合法性检查[2,3]。freeRTOS提供的动态内存管理方案在动态内存回收时只是将其从小到大排列并不把相邻的两块空闲区合并成一个大空闲区，该方案将可能导致整个动态内存分区被分割成很多个细小的内存碎片，同样，freeRTOS不对回收的内存块进行合法性检查[4,5]。

针对上述存在的不足，本文提出一种多平台通用、高效率、简单易用同时具有动态内存溢出检测机制的微型嵌入式动态内存管理机制HeapManager。

2 HeapManager的实现(Realization of HeapManager)

本文基于现有嵌入式操作系统动态内存管理机制的原理，研究和分析了多个微型嵌入式操作系统内存管理的优缺点，以多平台通用、高效率、简单易用为目标，实现了一种新的动态内存管理机制方法HeapManager。

2.1 数据结构

HeapManager将动态内存区连续内存空间视为一块空闲分块，并为每一块空闲块建立一个链表节点以记录该空闲块信息，将所有空闲块链表节点组成一个空闲分区链表，其链表节点的数据结构类型如下：

```
typedef struct BlockLink
{
    struct BlockLink *nextFreeBlock;        //下一个空间分区节点指针
    HeapSizeType  blockSize;                //空闲分区大小
}BlockLinkType;
```

其中，nextFreeBlock是指向下一个空闲分区节点指针；blockSize存放本节点代表的空闲分区大小，其数据类型为HeapSizeType。HeapSizeType根据用户实际应用时所设置的动态分区大小configTotalHeapSize确定(unsigned char、

基金项目：广州市科技和信息化局科普计划资助项目，“搭建原创智能车平台，培养中小学生工程实践意识”(项目编号：2013KP054)；广东省教育科研“十二五”规划2012年度研究项目，“构建职教师资人才能力培养开放性创新实验体系的探索与实践”(项目编号：2012JK045).

unsigned int或unsigned long)，实现代码如下：

```
//堆空间大小的数据类型
#if configTotalHeapSize > 65535
  typedef unsigned long HeapSizeType;
#elif configTotalHeapSize > 255
  typedef unsigned int HeapSizeType;
#else
  typedef unsigned char  HeapSizeType;
#endif
```

HeapManager提供动态内存系统的错误检查，当用户程序调用接口函数(heapMalloc和heapFree)执行动态内存的申请或释放时，HeapManager会根据执行的过程将结果信息反馈给用户程序，返回结果定义如下：

```
#define NO_ERR_HEAP              0       //没有错误
#define NO_FREE_BLOCK_HEAP       1       //没有适合的空闲空间
#define BLOCK_SIZE_ERR           2       //内存块大小错误
#define BLOCK_ADDR_ERR           3       //所释放内存块地址错误
#define BLOCK_STRUCT_DESTORY     4       //内存块结构损坏
#define UNKNOWN_ERROR            255     //未知错误
```

2.2 算法实现

HeapManager对用户程序提供三个接口函数：heapMalloc()、heapFree()和heapGetFreeSize()，其中heapMalloc()用于用户程序申请动态内存，heapFree()用于用户程序使用完动态内存后释放回系统动态内存堆，heapGetFreeSize()用于取得当前系统动态内存堆中所有空闲空间的总大小。

void *heapMalloc(HeapSizeType wantedSize, unsigned char *err)是动态内存空间申请函数，当用户调用该函数时须传递两个参数：wantedSize表示“所需要的动态内存大小”，err表示“保存执行结果信息的指针”。该函数返回管理机制所分配的动态内存块首地址(void*)并将错误信息存入变量err(没有错误则为0)。

heapMalloc()会检查自身是否第一次被调用。在用户首次调用heapMalloc()时，heapMalloc()会执行一次初始化堆分区建立一个空闲分区链表。初始化的过程包括：建立整个空闲分区链表的起始节点和终止节点；将初始节点的下一个分区节点指向系统动态内存堆首地址，初始节点空闲分区大小定为0；将终止节点的下一个分区节点指向NULL，终止节点空闲分区大小定为整个系统动态内存堆长度；建立第一个空闲分区节点并存放于系统动态内存堆顶部，将该节点的下一个分区节点指向终止节点，定义该节点空闲分区大小为整个系统动态内存堆总长度减去节点所占空间。

heapMalloc()被调用后会对参数wantedSize进行检测判断，如果wantedSize不大于系统所剩余空闲空间，则从heapStart指向的空闲空间链表开始对比每个空闲分区的大小是否大于所申请长度，直到找到第一个合适的空闲分区或者heapEnd结束，再根据除去用户所申请空间的空闲分区大小决定是否重新创建一个新空闲分区节点，最后更新空闲分区链表并返回所申请内存(保留顶部节点信息)的首地址。

unsigned char heapFree(void*freeBlock)在用户程序使用完所申请内存空间后释放该动态内存时被调用，用户调用时须传递参数freeBlock指向所释放内存的首地址，函数结束后将返回执行结果信息(没有错误返回0)。

heapFree()被调用后会根据freeBlock指向的地址得到回收空闲分区的空闲分区节点并进行检测，比较freeBlock所指向的位置是否包含于动态内存区，如果不是则返回错误，再判断该空闲分区节点结构是否损坏，然后从空闲分区链表heapStart所指向的第一个空闲分区开始地址对比，找到该内存块合适的回收位置，进行回收操作，在HeapManager中会根据回收空间所在位置的不同情况作不同的回收处理。

HeapSizeType heapGetFreeSize(void)函数被调用时返回HeapManager中所剩空闲分区的总大小，该大小为绝对空闲区，即除去空闲分区节点所占用后剩下的空间。如果在第一次调用heapMalloc()函数前调用heapGetFreeSize()则返回用户所设置的总动分区大小configTotalHeapSize。

2.3 设置与使用

HeapManager的定位是一个简单易用的微型管理机制，所以在使用HeapManager时需要设置的参数仅有两个，分别是_HEAP_MANAGER_ON_和configTotalHeapSize。

_HEAP_MANAGER_ON_用于定义是否启用HeapManager。我们将HeapManager的两个文件(C文件和H文件)添加到项目工程中，如果将_HEAP_MANAGER_ON_设置为0则编译器在编译时不会产生HeapManager代码，也即关闭HeapManager。configTotalHeapSize设置系统用于动态内存分区的总大小。开发人员在使用HeapManager之前须根据硬件资源为HeapManager划分一块configTotalHeapSize大小的内存区作为动态内存分区，该值必须大于一个空闲分区节点的长度(空闲分区节点长度=平台数据指针所占空间+HeapSizeType实际表示空间)。HeapManager会在用户程序申请或释放动态内存空间时进行各种检测，如果出现错误则将该错误的信息反馈给用户程序，开发人员可根据

(下转第58页)