**扩展的内存管理机制设计实验**

1. **实验目的**

在了解实时嵌入式操作系统内存管理机制的特点以及实时处理对内存管理需求的基础上，练习并掌握有效处理内存碎片的内存管理机制，同时理解防止内存泄漏问题的良好设计方法。

1. **实验工具及环境**

Tornado、VxSim仿真环境。

1. **实验原理及要点**

实时多任务操作系统（如VxWorks）为保证其时实性，在内存管理方面往往采用最简单有效的方式进行（如malloc/free），但同时会辅以扩展功能（如内存分区-Memory Partition、Cache安全等）以满足用户对内存操作的基本需求。

但实践者应该认识到，在使用操作系统所提供的内存管理机制时，需注意防止其造成的潜在危险，需要形成良好的使用习惯，同时根据应用的特点适当扩展。其关键点在于：

1） 实时系统的内存管理的主要目标为：快速、高效、高可靠；

2） 内存碎片（外部碎片）是危害系统稳定的重要原因之一；

3） 预先规划是达到以上目标的一种良好手段。由于实时系统无法使用复杂的内存管理机制（如垃圾回收等），用户可以根据实时应用使用内存的特点，扩展自己私有的内存管理机制；

4） 内存泄漏的防治主要依赖于内存使用上严谨的编程习惯。

1. **实验内容**

1）使用预先规划的思想，构建自己的私有内存管理机制，在系统内存池中申请内存，并将其纳入私有内存管理机制中，形成静态预分配内存池；

2）静态预分配内存池支持一种以上固定长度内存池，如16字节内存池和256字节内存池。固定长度内存池的单块长度应考虑体系结构开销，并尽量减少内部碎片；固定长度内存池数量应可配置；

3）静态预分配内存池与系统内存池的统一管理机制。向用户分配内存时应保证长度最佳匹配原则。当申请内存的长度超过静态预分配长度或资源不足时，自动向系统内存池申请；

4）管理机制至少应包括：

a) 初始化函数；

b) 内存申请/释放函数。并特别要保证释放安全；

c) 管理监视机制。

5）利用可能的互斥机制或代码可重入设计，保证以上管理机制的操作安全性；

6）创建多Task环境测试及演示以上内容。

1. **实验设计**
   1. **系统结构图**

系统结构图，如图1所示，内存区域包含自定义的静态预分配内存池。此系统结构可以实现内存池数量、池中内存块的大小、数量等自定义操作，并且整个系统结构提供统一的API接口，供其他任务调用申请分配内存。静态预分配内存池采用内存控制块（Memory Control Block）进行统一管理，每个MCB具有其所管理的内存池的基本信息。

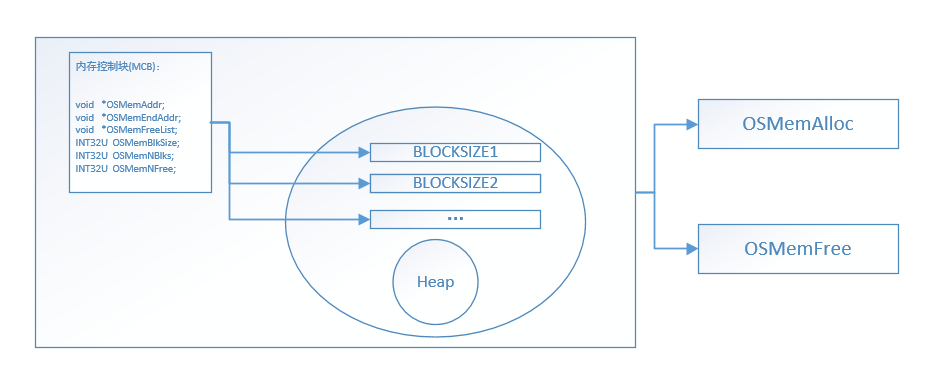


图1 系统结构图

* 1. **核心常量、结构和变量说明**

1）核心常量说明

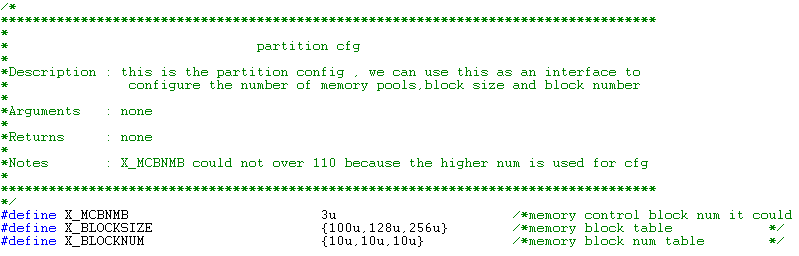


图2 核心常量

如图2所示，核心常量采用宏定义形式，共有三个，均为无符号整数，位于 x\_os\_cfg.h 文件中。X\_MCBNMB 表示内存控制块的数目，即静态预分配内存池的数量，X\_MCBNUM 所配置的数量不能超过110个，即，内存池数量最多可以设置110个。X\_BLOCKSIZE 此常量采用宏定义的形式，来初始化 X\_BLOCKSIZE\_TABLE 数组，表示每个内存池中的内存块的大小。X\_BLOCKNUM 此常量同样采用宏定义的形式，来初始化 X\_BLOCKNUM\_TABLE 数组，表示每个内存池中的内存块数目。有了如上所述的三个核心常量，则可以定义内存池的大小，即 MEMORY\_SIZE = BLOCKSIZE \* BLOCKNUM 。

2）核心结构说明



图3 核心结构体

如图3所示，核心结构体 X\_OS\_MEM 即内存控制块（MCB）共有6个数据成员，位于x\_os\_cfg.h 头文件中，共同来控制其所管理的内存池。OSMemAddr 表示内存池的起始地址，OSMemEndAddr 表示内存池的终止地址，OSMemFreeList 用来指示内存池内部内存块的链表地址，OSMemBlkSize 表示内存池中每个内存块的大小，OSMemNBlks 表示内存池所拥有的内存块的总个数，OSMemFree 表示内存池中当前可用的内存块的个数。

3）核心变量说明

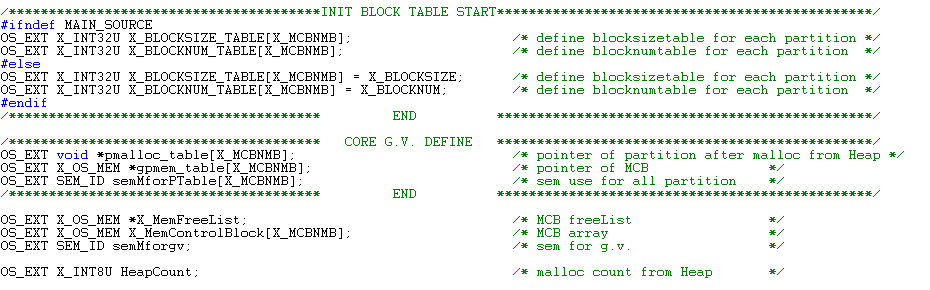


图4 核心变量

如图4所示，核心变量全部被设置为全局变量，位于 x\_os\_cfg.h 头文件中，对于变量X\_BLOCKSIZE\_TABLE 与 X\_BLOCKNUM\_TABLE 采用预编译的方式来区分其初始化与声明。X\_BLOCKSIZE\_TABLE 表示每个内存池中的内存块的大小，X\_BLOCKNUM\_TABLE 表示每个内存池中内存块的数目，分别用 X\_BLOCKSIZE 、X\_BLOCKNUM 两个核心常量来进行初始化工作。

pmalloc\_table 数组表示指向每个内存池从堆中划分出来的静态存储区的地址，用此数组来接收 malloc 的返回值。gpmem\_table 数组为指针数组，此数组中的每个元素都指向特定的内存控制块，其下标在模式选择的时候表示为其对应的模式，这样就可以实现随机访问MCB。semMforPTable 数组用来存取信号量的ID，因为此系统结构可以划分110个之内的内存池，虽然内存池对外提供统一的API，但是每个内存池之间并不是互斥的，即可以通过为每个内存池设置一个信号量，使得各个内存池之间可以实现同时访问，提高系统运行效率。

X\_MemFreeList 表示系统的空闲MCB列表，当需要创建一个内存池的时候，需要从链表上取下一个MCB，用以管理需要创建的的内存池。X\_MemControlBlock 表示MCB数组，作为全局变量，系统在初始化的时候为其分配内存，即MCB所占有的内存空间是预先静态申请好的。semMforgv 是为访问全局变量所设置的互斥信号量，以防止两个或以上的任务同时修改同一全局变量。HeapCount用来记录在内存分配过程，若静态预分配的内存池不满足条件，需要从堆里申请的任务数量。

* 1. **核心思想说明**

1. 内存池内部内存块的组织方式

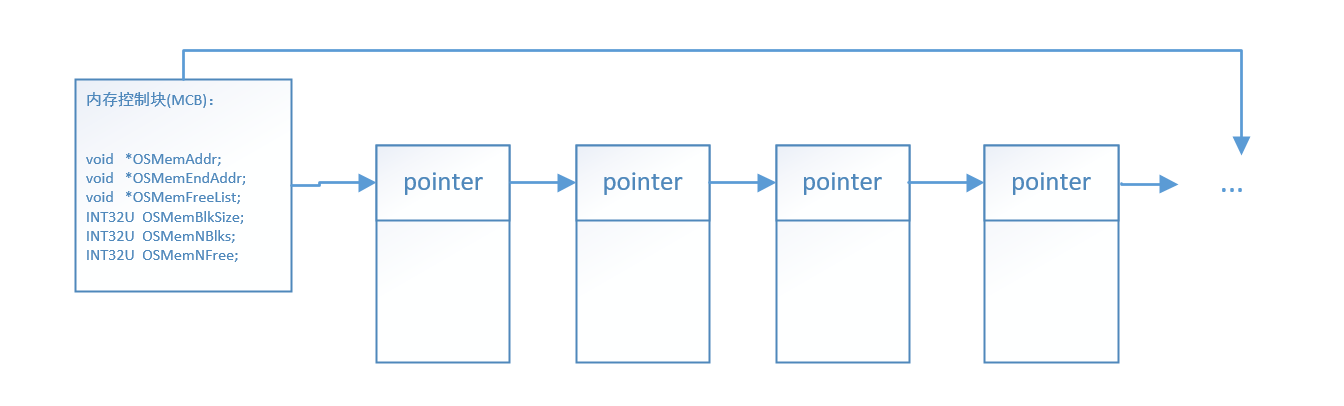


图5 内存块的组织方式

在内存池的内部，由于从堆中malloc申请到的内存是连续的，故可以借鉴 uC/OS II 内存管理的思想，在创建内存池的时候，将其内部的内存块以图5的方式组织起来，因为此时内存块并未被使用，其内部空间为空，故可以用来存储指向下一个内存块的指针，这也就同时要求，每个内存块的大小必须至少能够容纳一个指针，指向内存块链的第一块内存的地址为OSMemAddr指针，指向内存池的最后一个字节的下一个地址的指针（尾后指针）为OSMemEndAddr。

1. 获取和释放内存块

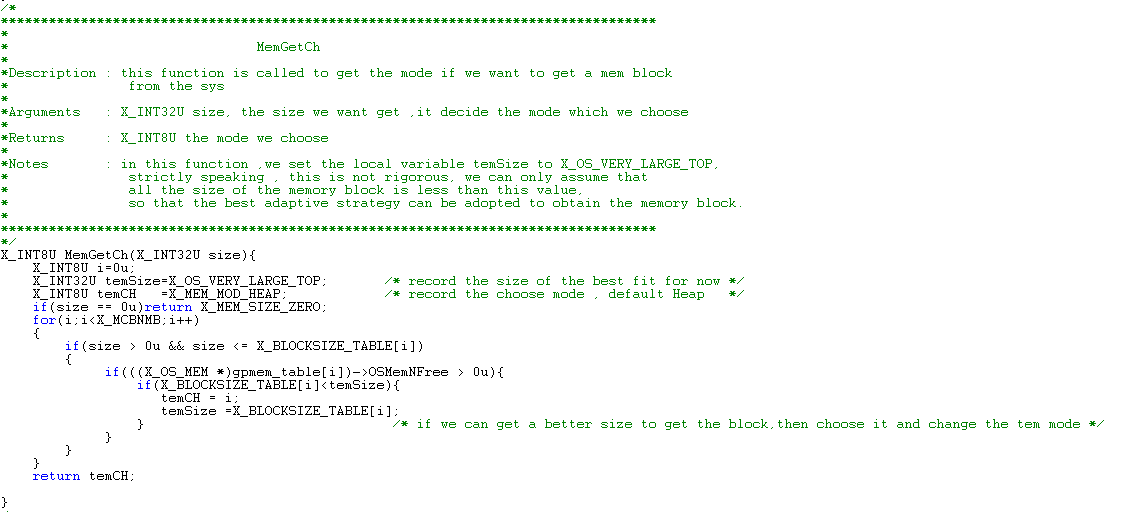


图6 最佳适应算法

如图6所示，获取内存块采用最佳适应算法，即总是在具有空闲内存块的内存池中选择大小最合适的内存池，来进行内存分配。

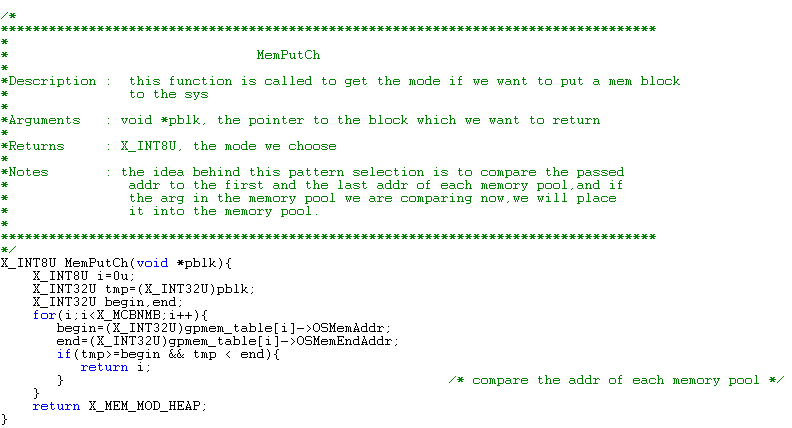


图7 释放内存块

如图7所示，释放内存块采用直接对比内存地址的方法，因为每个内存池在创建的时候都是单独被申请，故拥有独立的首尾地址空间，被记录在MCB中，所以可以直接对比得出内存块的地址在哪个内存池范围内，进而可以释放内存块。

3）监视机制

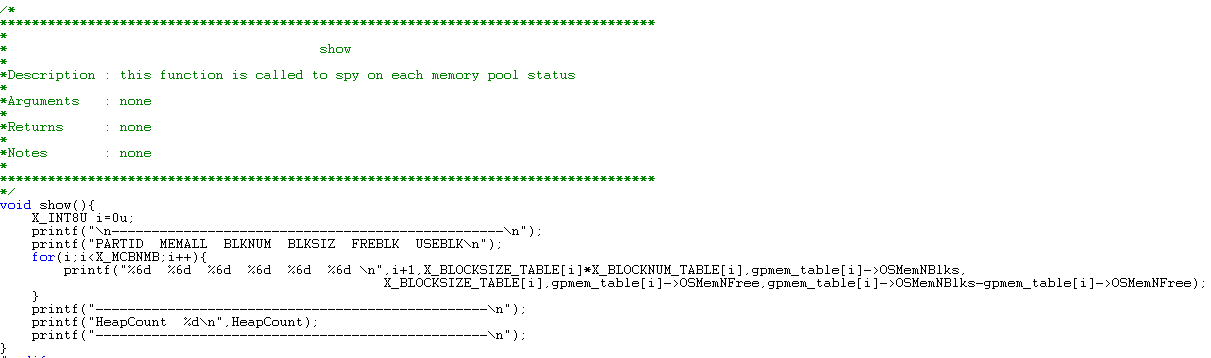


图8 监视函数show()

如图8所示，系统的内存状态主要由监视函数show()负责，show()函数可以打印输出当前系统的内存池状态，堆使用状态。内存池状态包括：内存池ID、内存池总大小、内存池中的内存块数量、内存块的大小、可用的内存块数量、已用的内存块数量，另外，采用全局变量HeapCount来记录从堆中申请内存的现有数量。

* 1. **主要过程说明**

系统主要具有三个过程：初始化过程、分配内存过程、释放内存过程。

1. 初始化过程

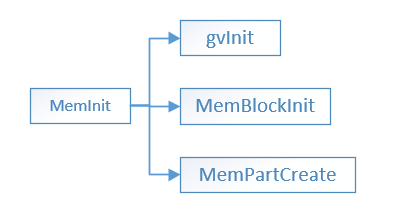


图9 初始化过程

如图9所示，初始化过程主要分为三步：

a）gvInit()：初始化全局变量，在此函数中，系统初始化所需的所有全局变量，并且创建互斥信号量，以及向系统堆内存中划分出静态预分配内存池，将malloc的返回值赋值给全局变量pmalloc\_table 。

b）MemBlockInit()：初始化内存控制块（MCB），在此函数中，系统按字节清空置0内存控制块区域，并且将内存控制块以链表的形式链接在一起，最后将内存控制块链的首地址赋值给全局变量 X\_MemFreeList 。

c）MemPartCreate()：初始化内存池，在此函数中，系统依次初始化每个内存池。对每个内存池系统都从全局变量 X\_MemFreeList 中摘下空闲的MCB，来管理内存池的状态，并且将内存池的内部内存块以图4的形式组织起来。

2）分配内存过程

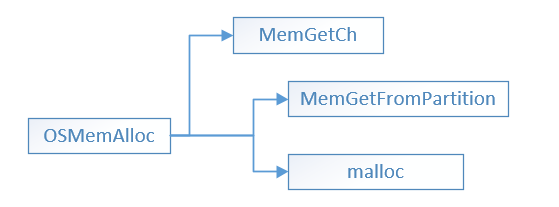


图10 分配内存过程

如图10所示，分配内存过程主要分为两步：

a）MemGetCh()：获取选择的内存池，在此函数中，系统根据需要分配的内存大小，采用最佳适应算法获取内存池的下标，若没有匹配的内存池，则返回从堆中获取的标识。

b）MemGetFromPartition()、malloc()：进行内存分配，在此步骤中，根据第一步所获得的返回值，来选择调用哪个函数，从而进行内存块的分配。

3）释放内存过程

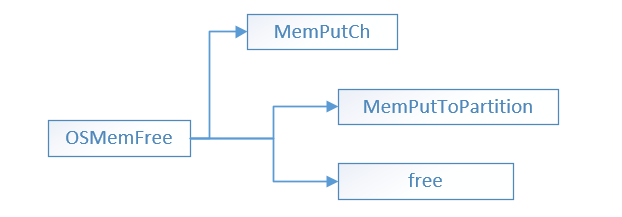


图11 释放内存过程

如图11所示，释放内存过程主要分为两步：

a）MemPutCh()：获取选择的内存池，在此函数中，系统根据需要释放的内存块的地址，来决定将内存块释放到哪一个内存池中，若无匹配内存池，则返回释放到堆的标识。

b）MemPutToPartition()、free()：进行内存释放，在此步骤中，根据第一步所返回的返回值，来选择调用那个函数，进而将内存块释放到指定位置。

1. **实验测试**

实验测试采用另一个独立的c文件来创建多Task环境申请内存，代码保存在test.c文件中。本实验共有三个测试：

1）测试边界值申请内存，范围内申请内存过程。核心代码：

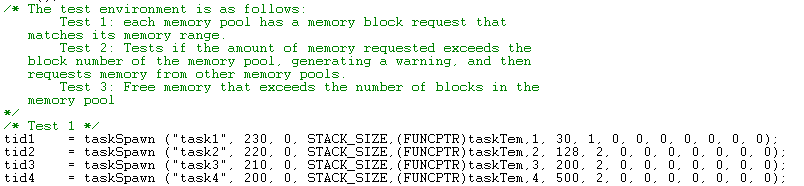


图12 测试一

如图12所示，系统预设三个内存池，分别为5个40字节的内存块、10个128字节的内存块、10个256字节的内存块。对于每个内存池，都有在其内存范围内的内存申请，包括边界值内存申请，来进行边界测试。task4申请500字节的内存，用来测试从堆中申请内存的过程。

测试结果：

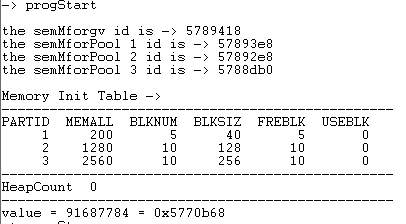


图13 Shell

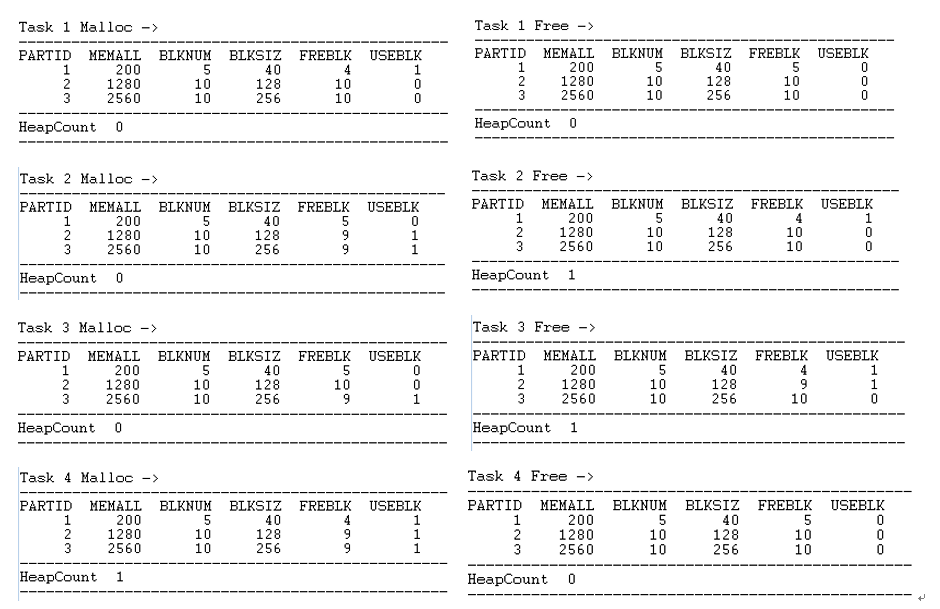


图14 Server

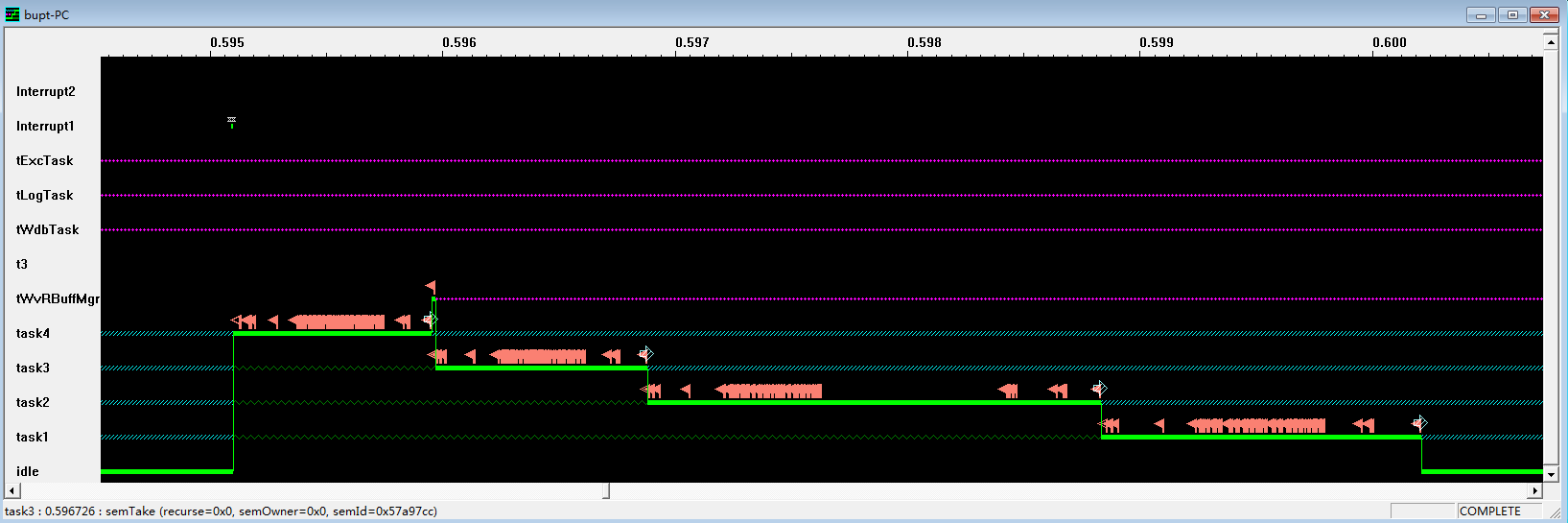


图15 Wind View

如图13、图14、图15所示，四个任务交替运行，在每个任务申请内存前，申请内存后，释放内存后，都会进行Task Delay。图13、图14展示了Shell端和Server端的输出情况，由打印出的表格可以看出，内存正确的被申请和释放。由图15所示，任务之间的切换，包括tWvRBuffMgr任务的抢占，都运行良好。

对图15中的任务task1运行进行详细分析，由于task1是申请内存池大小为40\*5字节的，所以它需要访问的信号量是内存池1的信号量，即信号量semId = 0x57893e8，放大task1段。

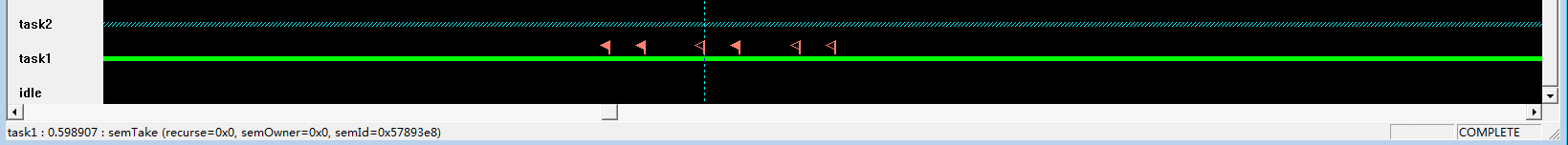


图16 Wind View中task1获得信号量

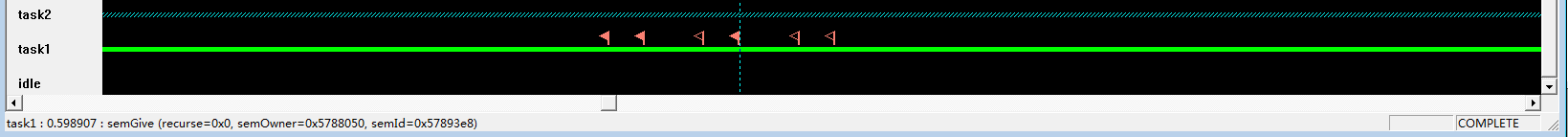


图17 Wind View中task1释放信号量

如图16、图17所示，展示了task1获取内存池1的信号量，释放内存池1的信号量的过程，从而展示了对内存池1的互斥操作。

2）测试向内存池申请过多的内存块。核心代码：

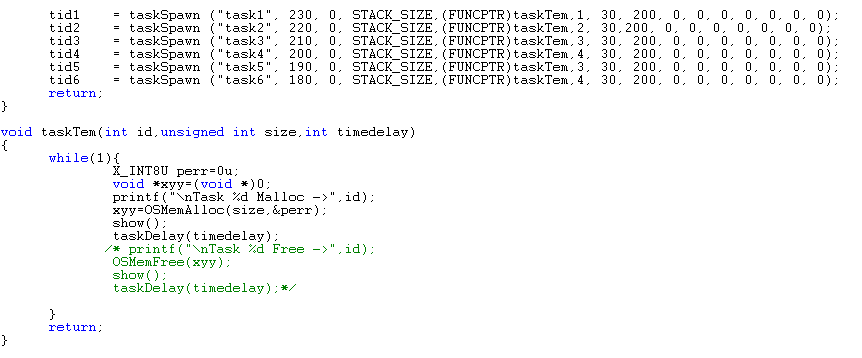


图18 申请过多内存块

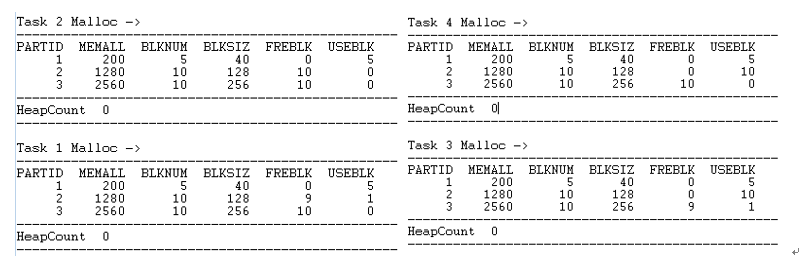


图19 申请内存块

如图18所示，申请内存池1的任务数量改为了6个，并且由taskTem()函数得知，任务只申请，并不会释放内存块，这样就会导致图19的结果，当内存池1的内存块数量不够的时候，会申请稍大一点的内存池2，当内存池2的内存块数量不够的时候，会申请稍大一点的内存池3 。可以看到，当申请过多的内存块的时候，系统不会崩溃，会一直选择当前情况下可用内存块大小最优的内存池来进行内存分配。

3）释放过多的内存块，核心代码：

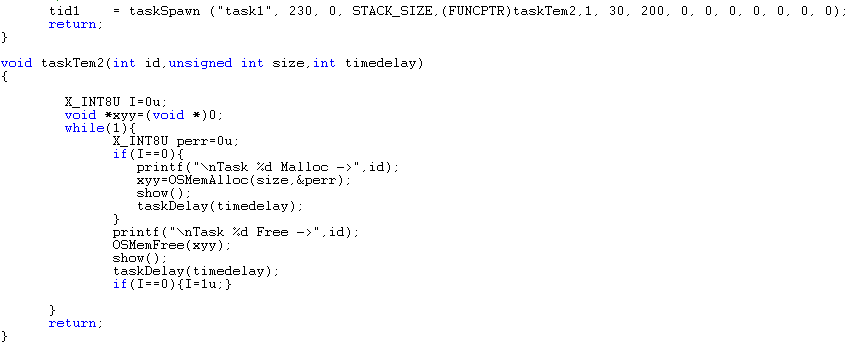


图20 核心代码

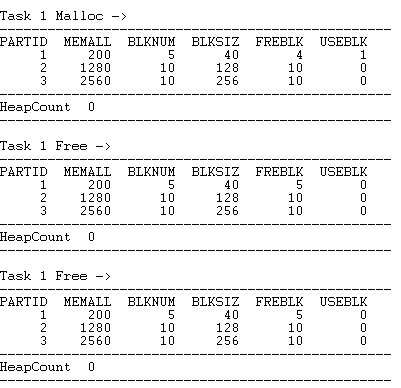


图21 释放过多内存块

如图20所示，用户仅仅创建了一个task，来申请或释放内存池1的内存块，定义了一个局部变量I来表示是否是第一次申请，因为需要第一次申请内存池1，来获得内存池1的内存地址，进而可以实现将内存释放到内存池1，在第二次循环的时候，I被置1，所以task不再进行申请内存操作，会一直释放内存块，结果如图21所示，即使释放过多内存块，系统仍然运行良好，不会出现错误。

1. **完整源代码**

1）x\_os\_cfg.h

#define CFGFILE /\* define this file for config \*/

#include "vxWorks.h"

#include "stdio.h"

#include "stdlib.h"

#include "semLib.h"

#include "taskLib.h"

#include "sysLib.h" /\* include the header file \*/

#ifdef OS\_GLOBALS

#define OS\_EXT

#else

#define OS\_EXT extern

#endif /\* global variable define \*/

#define X\_OS\_MEM\_EN 1u /\* enable memory management \*/

#define X\_OS\_ARG\_CHK\_EN 1u /\* enable arg check \*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* partition cfg

\*

\*Description : this is the partition config , we can use this as an interface to

\* configure the number of memory pools,block size and block number

\*

\*Arguments : none

\*

\*Returns : none

\*

\*Notes : X\_MCBNMB could not over 110 because the higher num is used for cfg

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#define X\_MCBNMB 3u /\*memory control block num it could not over 110 because the higher num is used for cfg\*/

#define X\_BLOCKSIZE {100u,128u,256u} /\*memory block table \*/

#define X\_BLOCKNUM {10u,10u,10u} /\*memory block num table \*/

/\* there are many modes ,and the mode allocated from the mem pool is [0,X\_MCBNMB) \*/

/\* if the mem pool could not afford the size we want,it will automatically convert to Heap mode \*/

#define X\_MEM\_MOD\_HEAP 111u /\* Heap mode \*/

#define X\_MEM\_SIZE\_ZERO 112u /\* malloc size is zero \*/

#define X\_MEM\_MOD\_RESERVE 113u /\* mode reserve tmp \*/

/\* sys bool variable \*/

#define X\_OS\_TRUE 0u /\* for true \*/

#define X\_OS\_FALSE 1u /\* for false \*/

/\* err code for create partition \*/

#define X\_OS\_ERR\_DIV\_NONE 0u /\* no err \*/

#define X\_OS\_ERR\_DIV\_MEM\_INVALID\_ADDR 1u /\* err for invalid addr arg \*/

#define X\_OS\_ERR\_DIV\_MEM\_INVALID\_BLKS 2u /\* err for blks num less 2 \*/

#define X\_OS\_ERR\_DIV\_MEM\_INVALID\_SIZE 3u /\* err for blks size too small \*/

#define X\_OS\_ERR\_DIV\_MEM\_INVALID\_PART 4u /\* MCB left zero \*/

#define PutErr

#define X\_OS\_PUT\_ERR\_NONE 0u /\* no err \*/

#define X\_OS\_ERR\_PUT\_NO\_MODE 1u /\* mode choose failed \*/

#define X\_OS\_ERR\_PUT\_INVALID\_MODE 2u /\* mode invalid \*/

#define X\_OS\_ERR\_PUT\_INVALID\_PBLK 3u /\* pblk invalid \*/

#define X\_OS\_ERR\_PUT\_MEM\_FULL 4u /\* partition full \*/

#define GetErr

#define X\_OS\_GET\_ERR\_NONE 0u /\* no err \*/

#define X\_OS\_ERR\_GET\_NO\_MODE 1u /\* mode choose failed \*/

#define X\_OS\_ERR\_GET\_INVALID\_MODE 2u /\* MODE invalid \*/

#define X\_OS\_ERR\_GET\_MALLOC\_FAIL 3u /\* malloc failed \*/

#define X\_OS\_ERR\_GET\_NO\_FREE\_BLKS 4u /\* no free blk in partition \*/

#define X\_OS\_VERY\_LARGE\_TOP 1000u /\* each block size should not over it \*/

typedef unsigned int X\_INT32U;

typedef unsigned char X\_INT8U;

#if X\_OS\_MEM\_EN > 0

typedef struct{

void \*OSMemAddr; /\* partition begin addr \*/

void \*OSMemEndAddr; /\* partition end addr \*/

void \*OSMemFreeList; /\* blks freeList \*/

X\_INT32U OSMemBlkSize; /\* blks size \*/

X\_INT32U OSMemNBlks; /\* blks num \*/

X\_INT32U OSMemNFree; /\* free blks num \*/

}X\_OS\_MEM;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INIT BLOCK TABLE START\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef MAIN\_SOURCE

OS\_EXT X\_INT32U X\_BLOCKSIZE\_TABLE[X\_MCBNMB]; /\* define blocksizetable for each partition \*/

OS\_EXT X\_INT32U X\_BLOCKNUM\_TABLE[X\_MCBNMB]; /\* define blocknumtable for each partition \*/

#else

OS\_EXT X\_INT32U X\_BLOCKSIZE\_TABLE[X\_MCBNMB] = X\_BLOCKSIZE; /\* define blocksizetable for each partition \*/

OS\_EXT X\_INT32U X\_BLOCKNUM\_TABLE[X\_MCBNMB] = X\_BLOCKNUM; /\* define blocknumtable for each partition \*/

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CORE G.V. DEFINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

OS\_EXT void \*pmalloc\_table[X\_MCBNMB]; /\* pointer of partition after malloc from Heap \*/

OS\_EXT X\_OS\_MEM \*gpmem\_table[X\_MCBNMB]; /\* pointer of MCB \*/

OS\_EXT SEM\_ID semMforPTable[X\_MCBNMB]; /\* sem use for all partition \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

OS\_EXT X\_OS\_MEM \*X\_MemFreeList; /\* MCB freeList \*/

OS\_EXT X\_OS\_MEM X\_MemControlBlock[X\_MCBNMB]; /\* MCB array \*/

OS\_EXT SEM\_ID semMforgv; /\* sem for g.v. \*/

OS\_EXT X\_INT8U HeapCount; /\* malloc count from Heap \*/

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

X\_INT8U gvInit();

void MemBlockInit();

X\_OS\_MEM \*MemCreate (void \*addr,X\_INT32U nblks,X\_INT32U blksize,X\_INT8U \*perr);

X\_INT8U MemPartCreate(X\_INT8U perr\_table[X\_MCBNMB]);

X\_INT8U MemGetCh(X\_INT32U size);

void \*MemGetFromPartition(X\_INT8U mode,X\_INT8U \*perr);

X\_INT8U MemPutCh(void \*pblk);

X\_INT8U MemPutToPartition(X\_INT8U mode,void \*pblk);

void MemInit();

void \*OSMemAlloc(X\_INT32U size,X\_INT8U \*perr);

X\_INT8U OSMemFree(void \*pblk);

void show();

2)x\_mem\_interface.c

#define OS\_GLOBALS

#define MAIN\_SOURCE

#include "x\_os\_cfg.h" /\* declaration of g.v. and #define \*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemInit

\*

\*Description : this function is called before the sys run,used for sys init

\*

\*Arguments : none

\*

\*Returns : none

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void MemInit(){

X\_INT8U i=0u;

X\_INT8U err=0u;

X\_INT8U perr\_table[X\_MCBNMB]={0};

err=gvInit(); /\* g.v. init \*/

if(err!=0)

{

printf("gvbInit gv failed \n");

return ;

}

MemBlockInit(); /\* MCB init \*/

err=0u;

err=MemPartCreate(perr\_table); /\* partition init \*/

if(err!=0u){

printf("mem div init failed \n");

for(i;i<X\_MCBNMB;i++){

printf("init mem %d for perr -> %d \n",i,perr\_table[i]);

}

return ;

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* OSMemAlloc

\*

\*Description : this function is used to get the mem from partitions or

\* Heap,the mode is dicided by the size passed by user

\*

\*Arguments : X\_INT32U size, the size we want to get

\* X\_INT8U \*perr, the err code

\*

\*Returns : (void \*)res pointer to the mem alloc,if an err occured,it will

\* return (void \*)0

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void \*OSMemAlloc(X\_INT32U size,X\_INT8U \*perr){

X\_INT8U mode=X\_MEM\_MOD\_RESERVE; /\* at the beginning.assign the mode to X\_MEM\_MOD\_RESERVE means no mode \*/

X\_INT8U err=0u;

void \*res=(void\*)0;

mode = MemGetCh(size);

if(mode==X\_MEM\_SIZE\_ZERO){ /\* if the size passed is 0 \*/

\*perr=X\_OS\_ERR\_GET\_NO\_MODE;

return (void \*)0;

}

else if(mode==X\_MEM\_MOD\_HEAP){ /\* the mode is Heap,it can be caused by many reason:1.the size is too lage 2. all of the pools is full\*/

res=malloc(size);

if(res==(void\*)0)\*perr=X\_OS\_ERR\_GET\_MALLOC\_FAIL;

else{

HeapCount++; /\* record the Heap used count \*/

\*perr=X\_OS\_GET\_ERR\_NONE;

}

return res;

}

else if(mode>=0 && mode<X\_MCBNMB){ /\* if it is a normal case, we will choose a mode between [0,X\_MCBNUM) \*/

res=MemGetFromPartition(mode,&err);

if(res==(void\*)0)\*perr=err;

else \*perr=X\_OS\_GET\_ERR\_NONE;

return res;

}else if(mode == X\_MEM\_MOD\_RESERVE){ /\* if the mode is not being changed , an err occured \*/

\*perr=X\_OS\_ERR\_GET\_NO\_MODE;

return (void \*)0;

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* OSMemFree

\*

\*Description : this function is called to free the mem we got before

\*

\*Arguments : void \*pblk pointer to the blk we want to free

\*

\*Returns : X\_INT8U err code

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_INT8U OSMemFree(void \*pblk){

X\_INT8U mode=X\_MEM\_MOD\_RESERVE; /\* at the beginning.assign the mode to X\_MEM\_MOD\_RESERVE means no mode \*/

X\_INT8U res=0u;

mode = MemPutCh(pblk); /\* choose the mode \*/

if(mode==X\_MEM\_MOD\_RESERVE)return X\_OS\_ERR\_PUT\_NO\_MODE; /\* if the mode is not being changed , an err occured \*/

else if(mode>=0 && mode<X\_MCBNMB){

res=MemPutToPartition(mode,pblk);

return res;

} /\* if it is a normal case, we will choose a mode between [0,X\_MCBNUM) \*/

else if (mode==X\_MEM\_MOD\_HEAP){ /\* if the mem we got is from Heap \*/

free(pblk);

HeapCount--; /\* record the Heap used count \*/

return X\_OS\_PUT\_ERR\_NONE;

}

}

3）x\_mem\_core.c

#include "x\_os\_cfg.h" /\* declaration for g.v. and #define \*/

#if X\_OS\_MEM\_EN >=1u

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* OS\_MemClr

\*

\*Description : this function is called to clear the mem by Byte

\*

\*Arguments : X\_INT8U \*pdest, the destination of mem we want to clear

\* X\_INT32U size, the range of the mem

\*

\*Returns : none

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void OS\_MemClr (X\_INT8U \*pdest,

X\_INT32U size)

{

while (size > 0u) {

\*pdest++ = (X\_INT8U)0; /\* clear the mem by Byte \*/

size--;

}

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* gvInit

\*

\*Description : this function is called to init the g.v. and malloc the mem of

\* partitions from Heap

\*

\*Arguments : none

\*

\*Returns : X\_INT8U err code

\* X\_OS\_FALSE : false

\* X\_OS\_TRUE : true

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_INT8U gvInit(){

X\_INT8U i=0u;

X\_MemFreeList = (void \*)0;

HeapCount = 0u; /\* init HeapCount for 0u \*/

semMforgv = semMCreate(SEM\_Q\_PRIORITY); /\* init the sem for g.v. ,the sem is Mutex \*/

printf("\nthe semMforgv id is -> %x",(X\_INT32U)semMforgv);

for(i;i<X\_MCBNMB;i++){

gpmem\_table[i] = (void \*)0; /\* assign the pointer (void \*)0 for now for temporary \*/

semMforPTable[i] = semMCreate(SEM\_Q\_PRIORITY); /\* partitions sem create \*/

printf("\nthe semMforPool %d id is -> %x",i+1,(X\_INT32U)semMforPTable[i]);

pmalloc\_table[i] = malloc(X\_BLOCKSIZE\_TABLE[i]\*X\_BLOCKNUM\_TABLE[i]);/\* malloc mem for each partition \*/

if( pmalloc\_table[i] == (void \*)0 ){

return X\_OS\_FALSE; /\* if an err occur,return immediately \*/

}

}

return X\_OS\_TRUE; /\* return the err code \*/

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemBlockInit

\*

\*Description : this function is called to initializes the memory control block and

\* links the MCB array into a linked list

\*

\*Arguments : none

\*

\*Returns : none

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void MemBlockInit(){

#if X\_MCBNMB >= 2u /\* the sys require 2 MCB at least \*/

X\_OS\_MEM \*pmem;

X\_INT8U i;

OS\_MemClr((X\_INT8U \*)&X\_MemControlBlock[0], sizeof(X\_MemControlBlock)); /\* clear the mem by byte \*/

for (i = 0u; i < (X\_MCBNMB - 1u); i++) {

pmem = &X\_MemControlBlock[i];

pmem->OSMemFreeList = (void \*)&X\_MemControlBlock[i + 1u];

} /\* link the array into list \*/

pmem = &X\_MemControlBlock[i];

pmem->OSMemFreeList = (void \*)0;

X\_MemFreeList = &X\_MemControlBlock[0]; /\* assign a true value to the g.v. X\_MemFreeList \*/

#endif

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemCreate

\*

\*Description : this function is called to initializes the memory partition and links

\* the blcoks into a linked list

\*

\*Arguments : void \*addr, the beginning addr of the partition

\* X\_INT32U nblks, the block num we want to create in the partiton

\* X\_INT32U blksize, the block size for each block

\* X\_INT8U \*perr, the err code

\*

\*Returns : X\_OS\_MEM \* , return pointer to the MCB which control the partition we

\* just create

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_OS\_MEM \*MemCreate (void \*addr,

X\_INT32U nblks,

X\_INT32U blksize,

X\_INT8U \*perr)

{

X\_OS\_MEM \*pmem; /\* pointer to the mcb \*/

X\_INT8U \*pblk; /\* pointer to the block \*/

void \*\*plink; /\* the second pointer to the block \*/

X\_INT32U loops;

X\_INT32U i;

#if X\_OS\_ARG\_CHK\_EN > 0u

if (addr == (void \*)0) {

\*perr = X\_OS\_ERR\_DIV\_MEM\_INVALID\_ADDR;

return ((X\_OS\_MEM \*)0);

}

if (nblks < 2u) {

\*perr = X\_OS\_ERR\_DIV\_MEM\_INVALID\_BLKS;

return ((X\_OS\_MEM \*)0);

}

if (blksize < sizeof(void \*)) {

\*perr = X\_OS\_ERR\_DIV\_MEM\_INVALID\_SIZE;

return ((X\_OS\_MEM \*)0);

}

#endif /\* check the arg we passed \*/

semTake(semMforgv, WAIT\_FOREVER); /\* we must take the sem in order to get the g.v. \*/

pmem = X\_MemFreeList;

if (X\_MemFreeList != (X\_OS\_MEM \*)0) {

X\_MemFreeList = (X\_OS\_MEM \*)X\_MemFreeList->OSMemFreeList;

} /\* we get a free MCB from the g.v. X\_MemFreeList \*/

semGive(semMforgv); /\* give the sem \*/

if (pmem == (X\_OS\_MEM \*)0) {

\*perr = X\_OS\_ERR\_DIV\_MEM\_INVALID\_PART;

return ((X\_OS\_MEM \*)0);

} /\* check the mcb pointer to prevent the pointer is null \*/

plink = (void \*\*)addr;

pblk = (X\_INT8U \*)addr;

loops = nblks - 1u;

for (i = 0u; i < loops; i++) {

pblk += blksize;

\*plink = (void \*)pblk;

plink = (void \*\*)pblk;

} /\* link the blocks into a link list \*/

pmem->OSMemAddr = addr;

pmem->OSMemEndAddr = (void \*)plink + blksize;

pmem->OSMemFreeList = addr;

pmem->OSMemNFree = nblks;

pmem->OSMemNBlks = nblks;

pmem->OSMemBlkSize = blksize; /\* fill in the corresponding value in the MCB \*/

\*plink = (void \*)0; /\* the last block point to null \*/

\*perr = X\_OS\_ERR\_DIV\_NONE;

return (pmem);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemPartCreate

\*

\*Description : this function is called to initializes the pointer array to the MCB

\*

\*Arguments : X\_INT8U perr\_table[X\_MCBNMB] ,err code

\*

\*Returns : X\_INT8U , the status of memory create

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_INT8U MemPartCreate(X\_INT8U perr\_table[X\_MCBNMB]){

X\_INT8U i=0u;

X\_INT8U flag=0u;

for(i;i<X\_MCBNMB;i++){

gpmem\_table[i] = MemCreate((void \*)pmalloc\_table[i],X\_BLOCKNUM\_TABLE[i],X\_BLOCKSIZE\_TABLE[i],&perr\_table[i]);

if(perr\_table[i]!=X\_OS\_ERR\_DIV\_NONE)flag=1u;

} /\* we create the partition one by one \*/

if(flag==1u)return X\_OS\_FALSE;

return X\_OS\_TRUE; /\* return the err code \*/

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemGetCh

\*

\*Description : this function is called to get the mode if we want to get a mem block

\* from the sys

\*

\*Arguments : X\_INT32U size, the size we want get ,it decide the mode which we choose

\*

\*Returns : X\_INT8U the mode we choose

\*

\*Notes : in this function ,we set the local variable temSize to X\_OS\_VERY\_LARGE\_TOP,

\* strictly speaking , this is not rigorous, we can only assume that

\* all the size of the memory block is less than this value,

\* so that the best adaptive strategy can be adopted to obtain the memory block.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_INT8U MemGetCh(X\_INT32U size){

X\_INT8U i=0u;

X\_INT32U temSize=X\_OS\_VERY\_LARGE\_TOP; /\* record the size of the best fit for now \*/

X\_INT8U temCH =X\_MEM\_MOD\_HEAP; /\* record the choose mode , default Heap \*/

if(size == 0u)return X\_MEM\_SIZE\_ZERO;

for(i;i<X\_MCBNMB;i++)

{

if(size > 0u && size <= X\_BLOCKSIZE\_TABLE[i])

{

if(((X\_OS\_MEM \*)gpmem\_table[i])->OSMemNFree > 0u){

if(X\_BLOCKSIZE\_TABLE[i]<temSize){

temCH = i;

temSize =X\_BLOCKSIZE\_TABLE[i];

} /\* if we can get a better size to get the block,then choose it and change the tem mode \*/

}

}

}

return temCH;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemGetFromPartition

\*

\*Description : this function is called to get the real mem block from the partition

\*

\*Arguments : X\_INT8U mode,the mode we choose before

\* X\_INT8U \*perr ,err code

\*

\*Returns : void \* ,the pointer of the mem block we get from the partition

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void \*MemGetFromPartition(X\_INT8U mode,X\_INT8U \*perr){

void \*pblk;

#if OS\_ARG\_CHK\_EN > 0u

if (mode >=X\_MCBNMB && mode !=X\_MEM\_MOD\_HEAP) {

return X\_OS\_ERR\_GET\_INVALID\_MODE;

}

#endif /\* check the arg we passed \*/

X\_OS\_MEM \*pmem = gpmem\_table[mode]; /\* get the pointer of the MCB \*/

semTake(semMforPTable[mode],WAIT\_FOREVER); /\* get the partition sem \*/

if (pmem->OSMemNFree > 0u) { /\* check the block num \*/

pblk = pmem->OSMemFreeList;

pmem->OSMemFreeList = \*(void \*\*)pblk;

pmem->OSMemNFree--;

semGive(semMforPTable[mode]);

\*perr = X\_OS\_GET\_ERR\_NONE;

return (pblk);

} /\* pick down a block from the block list and return the pointer \*/

semGive(semMforPTable[mode]);

\*perr = X\_OS\_ERR\_GET\_NO\_FREE\_BLKS;

return ((void \*)0);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemPutCh

\*

\*Description : this function is called to get the mode if we want to put a mem block

\* to the sys

\*

\*Arguments : void \*pblk, the pointer to the block which we want to return

\*

\*Returns : X\_INT8U, the mode we choose

\*

\*Notes : the idea behind this pattern selection is to compare the passed

\* addr to the first and the last addr of each memory pool,and if

\* the arg in the memory pool we are comparing now,we will place

\* it into the memory pool.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_INT8U MemPutCh(void \*pblk){

X\_INT8U i=0u;

X\_INT32U tmp=(X\_INT32U)pblk;

X\_INT32U begin,end;

for(i;i<X\_MCBNMB;i++){

begin=(X\_INT32U)gpmem\_table[i]->OSMemAddr;

end=(X\_INT32U)gpmem\_table[i]->OSMemEndAddr;

if(tmp>=begin && tmp < end){

return i;

} /\* compare the addr of each memory pool \*/

}

return X\_MEM\_MOD\_HEAP;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* MemPutToPartition

\*

\*Description : this function is called to return the block to the partition

\*

\*Arguments : X\_INT8U mode, the mode we choose before

\* void \*pblk, the pointer of the block we want to return

\*

\*Returns : X\_INT8U,the err code

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

X\_INT8U MemPutToPartition(X\_INT8U mode,void \*pblk){

#if OS\_ARG\_CHK\_EN > 0u

if (mode >=X\_MCBNMB && mode !=X\_MEM\_MOD\_HEAP) {

return X\_OS\_ERR\_PUT\_INVALID\_MODE;

}

if (pblk == (void \*)0) {

return (X\_OS\_ERR\_PUT\_INVALID\_PBLK);

}

#endif /\* check the arg we passed in \*/

X\_OS\_MEM \*pmem = gpmem\_table[mode]; /\* get the MCB pointer \*/

semTake(semMforPTable[mode], WAIT\_FOREVER); /\* get the sem for the partition \*/

if (pmem->OSMemNFree >= pmem->OSMemNBlks) {

semGive(semMforPTable[mode]);

return (X\_OS\_ERR\_PUT\_MEM\_FULL);

} /\* if the memory pool is full ,an error occured \*/

\*(void \*\*)pblk = pmem->OSMemFreeList;

pmem->OSMemFreeList = pblk;

pmem->OSMemNFree++;

semGive(semMforPTable[mode]); /\* return the block successfully \*/

return (X\_OS\_PUT\_ERR\_NONE);

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* show

\*

\*Description : this function is called to spy on each memory pool status

\*

\*Arguments : none

\*

\*Returns : none

\*

\*Notes : none

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void show(){

X\_INT8U i=0u;

printf("\n-------------------------------------------------\n");

printf("PARTID MEMALL BLKNUM BLKSIZ FREBLK USEBLK\n");

for(i;i<X\_MCBNMB;i++){

printf("%6d %6d %6d %6d %6d %6d \n",i+1,X\_BLOCKSIZE\_TABLE[i]\*X\_BLOCKNUM\_TABLE[i],gpmem\_table[i]->OSMemNBlks,X\_BLOCKSIZE\_TABLE[i],gpmem\_table[i]->OSMemNFree,gpmem\_table[i]->OSMemNBlks-gpmem\_table[i]->OSMemNFree);

}

printf("-------------------------------------------------\n");

printf("HeapCount %d\n",HeapCount);

printf("-------------------------------------------------\n");

}

#endif

4) test.c

#include "x\_os\_cfg.h"

#define STACK\_SIZE 2000

int tid1;

/\*int tid2;

int tid3;

int tid4;

int tid5;

int tid6;\*/

void progStart(void);

void taskTem(int id,unsigned int size,int timedelay);

void taskTem2(int id,unsigned int size,int timedelay);

void progStop(void);

void progStart(void)

{

MemInit();

printf("\n\nMemory Init Table ->");

show();

/\* The test environment is as follows:

Test 1: each memory pool has a memory block request that

matches its memory range.

Test 2: Tests if the amount of memory requested exceeds the

block number of the memory pool, generating a warning, and then

requests memory from other memory pools.

Test 3: Free memory that exceeds the number of blocks in the

memory pool

\*/

/\* Test 1

tid1 = taskSpawn ("task1", 230, 0, STACK\_SIZE,(FUNCPTR)taskTem,1, 30, 1, 0, 0, 0, 0, 0, 0, 0);

tid2 = taskSpawn ("task2", 220, 0, STACK\_SIZE,(FUNCPTR)taskTem,2, 128, 2, 0, 0, 0, 0, 0, 0, 0);

tid3 = taskSpawn ("task3", 210, 0, STACK\_SIZE,(FUNCPTR)taskTem,3, 200, 2, 0, 0, 0, 0, 0, 0, 0);

tid4 = taskSpawn ("task4", 200, 0, STACK\_SIZE,(FUNCPTR)taskTem,4, 500, 2, 0, 0, 0, 0, 0, 0, 0);

\*/

/\* Test 2

tid1 = taskSpawn ("task1", 230, 0, STACK\_SIZE,(FUNCPTR)taskTem,1, 30, 200, 0, 0, 0, 0, 0, 0, 0);

tid2 = taskSpawn ("task2", 220, 0, STACK\_SIZE,(FUNCPTR)taskTem,2, 30,200, 0, 0, 0, 0, 0, 0, 0);

tid3 = taskSpawn ("task3", 210, 0, STACK\_SIZE,(FUNCPTR)taskTem,3, 30, 200, 0, 0, 0, 0, 0, 0, 0);

tid4 = taskSpawn ("task4", 200, 0, STACK\_SIZE,(FUNCPTR)taskTem,4, 30, 200, 0, 0, 0, 0, 0, 0, 0);

tid5 = taskSpawn ("task5", 190, 0, STACK\_SIZE,(FUNCPTR)taskTem,3, 30, 200, 0, 0, 0, 0, 0, 0, 0);

tid6 = taskSpawn ("task6", 180, 0, STACK\_SIZE,(FUNCPTR)taskTem,4, 30, 200, 0, 0, 0, 0, 0, 0, 0);

\*/

tid1 = taskSpawn ("task1", 230, 0, STACK\_SIZE,(FUNCPTR)taskTem2,1, 30, 200, 0, 0, 0, 0, 0, 0, 0);

return;

}

void taskTem2(int id,unsigned int size,int timedelay)

{

X\_INT8U I=0u;

void \*xyy=(void \*)0;

while(1){

X\_INT8U perr=0u;

if(I==0){

printf("\nTask %d Malloc ->",id);

xyy=OSMemAlloc(size,&perr); /\* alloc mem from sys \*/

show();

taskDelay(timedelay);

}

printf("\nTask %d Free ->",id);

OSMemFree(xyy); /\* free mem to sys \*/

show();

taskDelay(timedelay);

if(I==0){I=1u;}

}

return;

}

void taskTem(int id,unsigned int size,int timedelay)

{

while(1){

X\_INT8U perr=0u;

void \*xyy=(void \*)0;

printf("\nTask %d Malloc ->",id);

xyy=OSMemAlloc(size,&perr); /\* alloc mem from sys \*/

show();

taskDelay(timedelay);

printf("\nTask %d Free ->",id);

OSMemFree(xyy); /\* free mem to sys \*/

show();

taskDelay(timedelay);

}

return;

}

void progStop(void)

{

taskDelete(tid1);

/\* taskDelete(tid2);

taskDelete(tid3);

taskDelete(tid4);

taskDelete(tid5);

taskDelete(tid6);\*/

printf("The End\n");

return;

}