静态内存管理之内存池

RealTouch 评估板 RT-Thread 入门文档

版本号：1.0.0

日期：2012/8/13

修订记录

日期	作者	修订历史
2012/8/13	bloom5	创建文档

实验目的

- 快速熟悉了解静态内存池相关的背景知识，API 以及相关概念

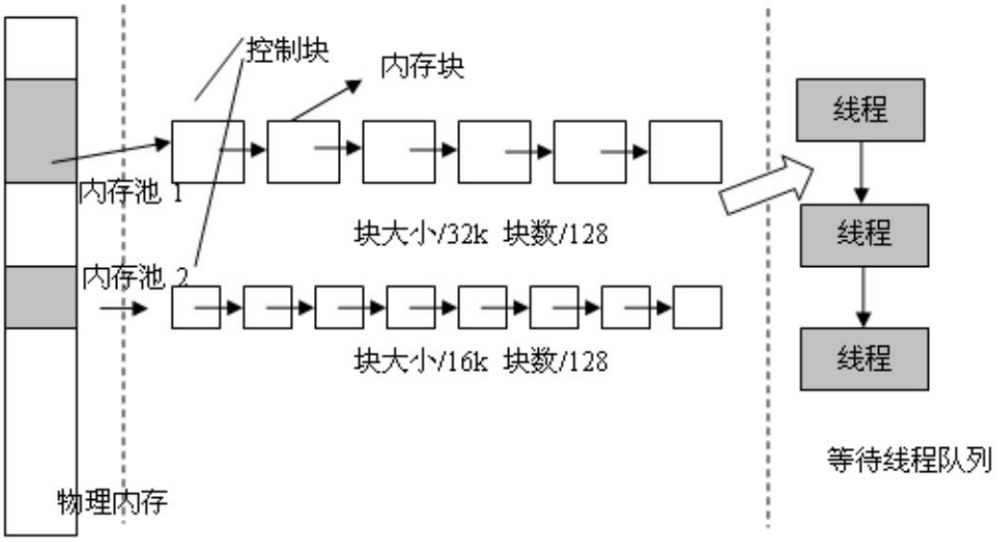
硬件说明

本实验使用 RT-Thread 官方的 Realtouch 开发板作为实验平台。涉及到的硬件主要为

- 串口 3，作为 rt_kprintf 输出，需要连接 JTAG 扩展板

具体请参见《Realtouch 开发板使用手册》

实验原理及程序结构



内存池管理结构示意图

内存池管理是一种静态的内存管理方法，它将一块连续的内存区域划分成几个大小不同的内存池，每个内存池中有固定数目个相同大小的内存块。使用者使用内存池方式申请内存时需要指定内存块大小，然后内存池管理算法会从对应块大小的内存池链表上进行分配内存块，返还给使用者；

这种内存分配的好处在于没有内存碎片，能够充分地使用每一个内存块，缺点在于每一个内存块大小固定，不够灵活。

内存池方式的内存管理方法适用于需要频繁使用和销毁内存对象的场合。

实验设计

本实验的主要设计目的是帮助读者了解内存管理中静态内存的分配和释放相关的 API。

源程序说明

本实验对应 1_kernel_mempool

系统依赖

在 rtconfig.h 中需要开启

- #define RT_USING_HEAP

此项可选，开启此项可以创建动态线程和动态信号量，如果使用静态线程和静态信号量，则此项不是必要的

- #define RT_USING_CONSOLE

此项必须，本实验使用 rt_kprintf 向串口打印按键信息，因此需要开启此项

主程序说明

我们首先定义了一个内存池的数据结构，和用于演示的两个线程

全局变量定义

```
static rt_uint8_t *ptr[48];
static rt_uint8_t mempool[4096];
static struct rt_mempool mp;

static rt_thread_t tid1 = RT_NULL;
static rt_thread_t tid2 = RT_NULL;
```

内存池的初始化及线程创建

```
int rt_application_init()
{
    int i;
    for (i = 0; i < 48; i ++) ptr[i] = RT_NULL;

    /* 初始化内存池对象 ,每块分配的大小为 80,但是另外还有大小为 4 的控制头，所以实际大小为 84*/
    rt_mp_init(&mp, "mp1", &mempool[0], sizeof(mempool), 80);

    /* 创建线程 1 */
    tid1 = rt_thread_create("t1",
            thread1_entry, RT_NULL, 512, 8, 10);
```

```
    if (tid1 != RT_NULL)
        rt_thread_startup(tid1);

    /* 创建线程 2 */
    tid2 = rt_thread_create("t2",
    thread2_entry, RT_NULL, 512, 9, 10);
    if (tid2 != RT_NULL)
        rt_thread_startup(tid2);

    return 0;
}
```

可以看到内存池的总大小也就是 sizeof(mempool),每个分块大小为 80。两个线程，tid1 优先级高于 tid2。

线程入口程序

```
static void thread1_entry(void* parameter)
{
    int i,j = 1;
    char *block;

    while(j--)
    {
        for (i = 0; i < 48; i++)
        {
            /* 申请内存块 */
            rt_kprintf("allocate No.%d\n", i);
            if (ptr[i] == RT_NULL)
            {
                ptr[i] = rt_mp_alloc(&mp, RT_WAITING_FOREVER);
            }
        }

        /* 继续申请一个内存块，因为已经没有内存块，线程应该被挂起 */
        block = rt_mp_alloc(&mp, RT_WAITING_FOREVER);
        rt_kprintf("allocate the block mem\n");
        /* 释放这个内存块 */
        rt_mp_free(block);
        block = RT_NULL;
    }
}

/* 线程 2 入口，线程 2 的优先级比线程 1 低，应该线程 1 先获得执行。*/
static void thread2_entry(void *parameter)
{
```

```
    int i,j = 1;

    while(j--)
    {
        rt_kprintf("try to release block\n");

        for (i = 0 ; i < 48; i ++)
        {
            /* 释放所有分配成功的内存块 */
            if (ptr[i] != RT_NULL)
            {
                rt_kprintf("release block %d\n", i);

                rt_mp_free(ptr[i]);
                ptr[i] = RT_NULL;
            }
        }

        /* 休眠 10 个 OS Tick */
        rt_thread_delay(10);
    }
}
```

编译调试及观察输出信息

编译请参见《RT-Thread 配置开发环境指南》完成编译烧录，参考《Realtouch 开发板使用手册》完成硬件连接，连接扩展板上的串口和 jlink。

运行后可以看到如下信息：

```
 \ | /
- RT -     Thread Operating System
 / | \     1.1.0 build Aug 10 2012
 2006 - 2012 Copyright by rt-thread team
allocate No.0
allocate No.1
allocate No.2
allocate No.3
allocate No.4
```

```
allocate No.5
allocate No.6
allocate No.7
allocate No.8
allocate No.9
allocate No.10
allocate No.11
allocate No.12
allocate No.13
allocate No.14
allocate No.15
allocate No.16
allocate No.17
allocate No.18
allocate No.19
allocate No.20
allocate No.21
allocate No.22
allocate No.23
allocate No.24
allocate No.25
allocate No.26
allocate No.27
allocate No.28
allocate No.29
allocate No.30
allocate No.31
allocate No.32
allocate No.33
allocate No.34
allocate No.35
allocate No.36
allocate No.37
allocate No.38
allocate No.39
allocate No.40
allocate No.41
allocate No.42
allocate No.43
allocate No.44
allocate No.45
allocate No.46
allocate No.47
try to release block
```

```
release block 0
allocate the block mem
release block 1
release block 2
release block 3
release block 4
release block 5
release block 6
release block 7
release block 8
release block 9
release block 10
release block 11
release block 12
release block 13
release block 14
release block 15
release block 16
release block 17
release block 18
release block 19
release block 20
release block 21
release block 22
release block 23
release block 24
release block 25
release block 26
release block 27
release block 28
release block 29
release block 30
release block 31
release block 32
release block 33
release block 34
release block 35
release block 36
release block 37
release block 38
release block 39
release block 40
release block 41
release block 42
```

```
release block 43
release block 44
release block 45
release block 46
release block 47
```

结果分析

本例程主要想要体现的就是内存池的初始化、分配以及释放，关于内存池相关更多 API 可以从编程指南上获得了解。

直接观察打印结果，可以看到在打印出 48 个 alloc 成功信息后，thread1 因为申请不到新的内存块而被挂起，thread2 获得控制权，打印出

```
try to release block
```

然后依次将由 thread1 中 48 个分得的内存块全部释放

```
for (i = 0 ; i < 48; i ++)
       {
           /* 释放所有分配成功的内存块 */
           if (ptr[i] != RT_NULL)
           {
               rt_kprintf("release block %d\n", i);

               rt_mp_free(ptr[i]);
               ptr[i] = RT_NULL;
           }
       }
```

在 thread2 释放完第一个内存块后，内存池中有了可用内存块，会将挂起在该内存池上的 thread1 线程唤醒，thread1 从而申请内存块成功，在 thread1 线程运行结束后，thread2 才继续完成剩余的内存块释放操作。正如实验结果所示。

此外，计算下 48 个内存块的总大小，结果是 80*48=3840 Bytes，而总的内存是 4096 Bytes，应该还是可以有两个内存块可供分配的，但为何实际缺分配不了呢，原因在于每个内存块都必须有一个控制头，这个控制头的大小为 4，这样计算一下，的确只能分配 48 块内存块，48 个内存块大小加上控制头大小的结果刚好是 4096 Bytes。