**课 程 设 计 报 告**

设计题目：实时操作系统内核

班 级：物联网1401

组长学号：20143862

组长姓名：唐在洋

指导教师：林树宽

设计时间：2017年7月

设计分工

组长学号及姓名： 20143862 唐在洋

分工：实时内核，时间管理

组员1学号及姓名：20143853 陈治铭

分工：内存管理

组员2学号及姓名：20143857 黄世华

分工：CPU移植相关

组员3学号及姓名：20143865 王清弘

分工：任务管理

**摘 要**

我们本次设计的是一个实时内核（我们自己取名叫EMOS）。是一个可以基于优先级调度、可裁剪的、抢占式、实时多任务内核，具有高度可移植性，特别适合于微处理器和控制器。其主要包含以下几大功能特征。

任务管理。包括创建、删除、挂起、唤醒等任务管理功能。在调度策略上采用优先级调度算法;时间管理。EMOS通过处理器的时钟确定一个系统时钟节拍，以此来标定系统当前的运行时间;内存管理。在EMOS中，并没有用C标准库的内存管理函数，而是采用每次分配或释放固定大小的内存块以保证确定的时间复杂度;高度可移植。EMOS采用ASIC C编程，并将所有与CPU架构、编译器相关的代码集中到一个C文件中，以方便移植修改;功能可配置。整个内核的功能模块可通过配置剪除不需要的功能模块代码，使生成的目标代码占用空间更小，以适应嵌入式设备有限的存储空间。

**关键词**：操作系统，实时调度，时间管理，任务管理，内存管理，调度策略，可移植性，EMOS

**目 录**

[**设计分工** I](#_Toc488241813)

[**摘 要** II](#_Toc488241814)

[**1 概述** 1](#_Toc488241815)

[**2 课程设计任务及要求** 2](#_Toc488241816)

[2.1 设计任务 2](#_Toc488241817)

[2.2 设计要求 2](#_Toc488241818)

[**3 算法与数据结构** 2](#_Toc488241819)

[3.1算法的总体思想（流程） 2](#_Toc488241820)

[3.2 实时内核 2](#_Toc488241821)

[3.2.1功能 2](#_Toc488241822)

[3.2.2数据结构 3](#_Toc488241823)

[3.2.3算法 5](#_Toc488241824)

[3.3 时间管理 7](#_Toc488241825)

[3.3.1功能 7](#_Toc488241826)

[3.3.2数据结构 7](#_Toc488241827)

[3.3.3算法 7](#_Toc488241828)

[3.4 内存管理 8](#_Toc488241829)

[3.4.1功能 8](#_Toc488241830)

[3.4.2数据结构 9](#_Toc488241831)

[3.4.3算法 9](#_Toc488241832)

[3.5任务管理 10](#_Toc488241833)

[3.5.1功能 10](#_Toc488241834)

[3.5.2数据结构 10](#_Toc488241835)

[3.5.3算法 10](#_Toc488241836)

[3.6 CPU移植相关 11](#_Toc488241837)

[3.6.1功能 11](#_Toc488241838)

[3.6.2数据结构 12](#_Toc488241839)

[3.6.3算法 12](#_Toc488241840)

[**4 程序设计与实现** 13](#_Toc488241841)

[4.1 程序流程图 13](#_Toc488241842)

[4.2 程序说明 20](#_Toc488241843)

[4.3 实验结果 20](#_Toc488241844)

[**5 结论** 22](#_Toc488241845)

[**6 参考文献** 23](#_Toc488241846)

[**7 收获、体会和建议** 23](#_Toc488241847)

**1 概述**

EMOS是一个可以基于优先级调度、可裁剪的、抢占式、实时多任务内核，具有高度可移植性，特别适合于微处理器和控制器。

任务管理。EMOS的任务管理功能包括创建任务、删除任务、挂起任务、唤醒任务。EMOS实时内核的任务调度策略是基于优先级的调度，首先为每一个任务确定一个优先级序号，并按小号高优先级，大号低优先级的规则在就绪表中进行排列，任务调度要做的就是从就绪队列中找出优先级最高的任务，将当前正在运行的任务与之交换CPU使用权。

时间管理。EMOS通过处理器的时钟确定一个系统时钟节拍，并在每个时钟节拍产生一个时钟中断，每来一个时钟节拍中断便会进入到相应的中断服务程序计数，以此来标定系统当前的运行时间，以及为每个任务计算任务超时时间值，每个任务都会有一个timout计数器，每个时钟中断都会在服务程序中将当前正在运行的任务超时计数器减1，如果timout的值等于0时，则当前任务将被剥夺CPU使用权，并切换到就绪队列中优先级最高的任务。以此来解决某些高优先级一直占用CPU而造成的调度不公平问题。

内存管理。EMOS的内存管理是按简单的固定块大小分配管理，对于一个实时内核，为了保证其在运行时间上的可预测性，每个动作必须有确定的时间复杂度，因此在EMOS中，并没有用C标准库的内存管理函数，而是采用每次分配或释放固定大小的内存块以保证确定的时间复杂度。

临界资源共享。在EMOS中通过信号量来解决临界资源的共享问题，系统会在启动前创建一个信号量并将初值设1，以后对于需要进行互斥访问的代码段，首先必须在入口处申请信号量，此时信号量值减1，如果信号量的值小于0，代表当前已经有任务正在访问当前资源，便阻塞当前任务，如果信号量等于0，则当前任务享有访问优先权。

高度可移植。为了提供最好的移植性能，EMOS最大程度上使用ANSI C语言进行开发，并将所有与CPU架构、编译器相关的代码集中到一个C文件中，移植到其它平台时只需要对一个C文件的代码做相应修改。

功能可配置。整个内核的功能模块可通过在内核配置文件中选择使能某些需要的功能进行重新编译，剪除不需要的功能模块代码，使生成的目标代码占用空间更小，以适应嵌入式设备有限的存储空间。

**2 课程设计任务及要求**

**2.1 设计任务**

在下列内容中任选其一：

1、多用户、多级目录结构文件系统的设计与实现；

2、WDM驱动程序开发；

3、存储管理系统的实现，主要包括虚拟存储管理调页、缺页统计等；

4、进程管理系统的实现，包括进程的创建、调度、通信、撤消等功能；

5、自选一个感兴趣的与操作系统有关的问题加以实现，要求难度相当。

在这次课程设计中，我们选择的题目是：5、实时操作系统内核

**2.2 设计要求**

1、在深入理解操作系统基本原理的基础上，对于选定的题目，以小组为单位，先确定设计方案；

2、设计系统的数据结构和程序结构，设计每个模块的处理流程。要求设计合理；

3、编程序实现系统，要求实现可视化的运行界面，界面应清楚地反映出系统的运行结果；

4、确定测试方案，选择测试用例，对系统进行测试；

5、运行系统并要通过验收，讲解运行结果，说明系统的特色和创新之处，并回答指导教师的提问；

6、提交课程设计报告。

**3 算法与数据结构**

**3.1算法的总体思想（流程）**

我们实现的实时内核在任务调度策略上主要采用静态优先级调度的算法来实现的；在查找就绪队列中最高优先级任务时采用的是位示图原理，在常数级的时间复杂度就能找到需要与当前任务进行切换的最高优先级的就绪任务；在时间管理上以系统节拍为基础，运用系统时钟节拍中断对任务运行时间进行度量；在内存管理上采用简单的固定大小块分配策略。

**3.2 实时内核**

3.2.1功能

完成任务调试的核心算法实现，对任务管理，内存管理，时间管理做数据结构以及算法实现上的支撑，是整个实时系统中向上服务的API级代码。主要包括：

**创建维护TCB链表；**

**任务控制块的初始化；**

**任务堆栈的创建与初始化；**

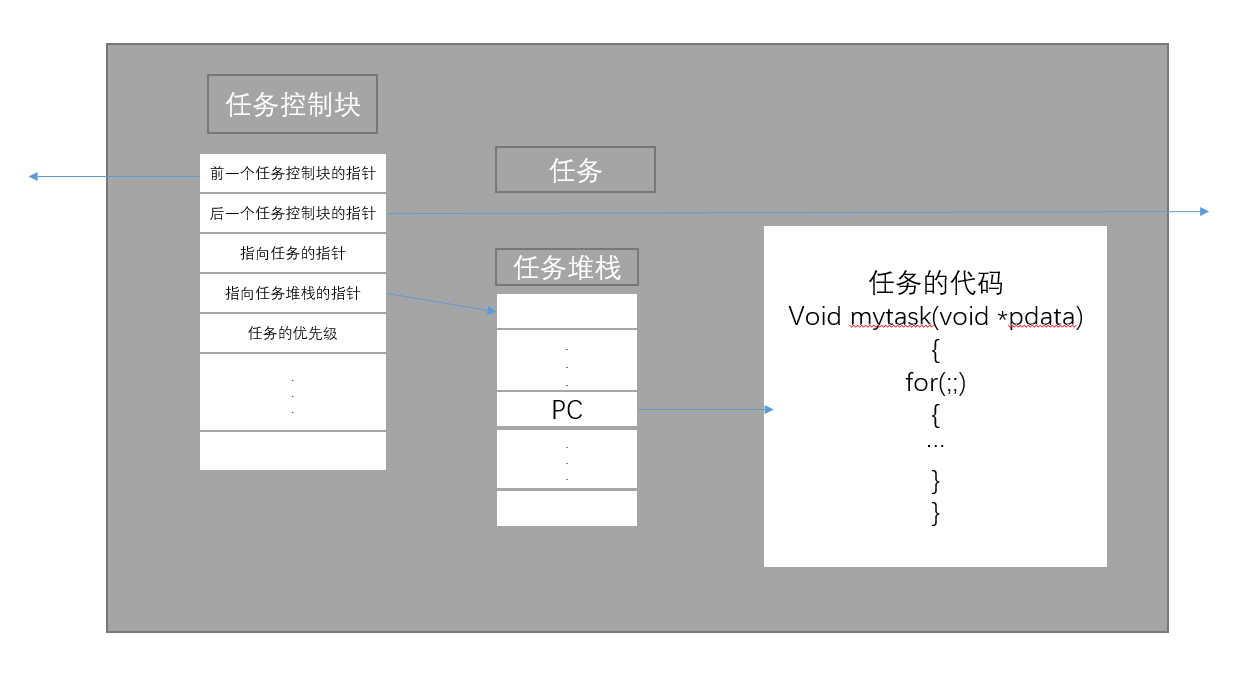
**系统级任务（空闲任务与统计任务）的实现；**

**创建并维护就绪表；**

**高优先级就绪任务的查找；**

3.2.2数据结构

1）任务控制块：

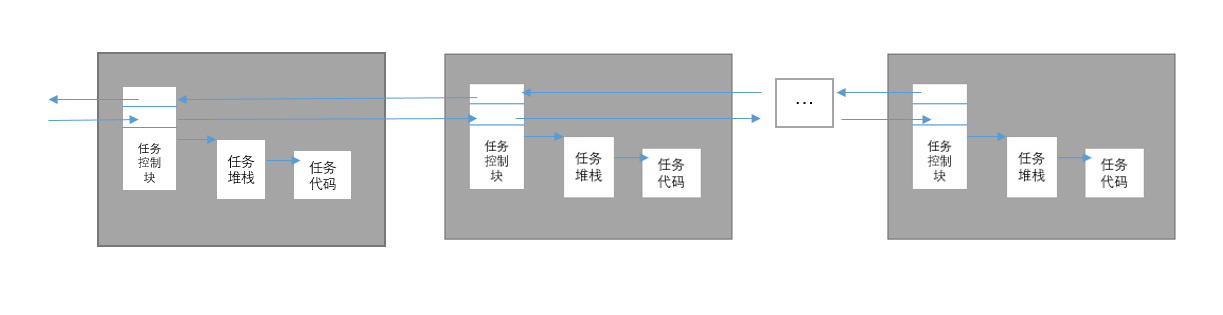


2）标识任务是否就绪的位示图（一共可以表示64个任务）

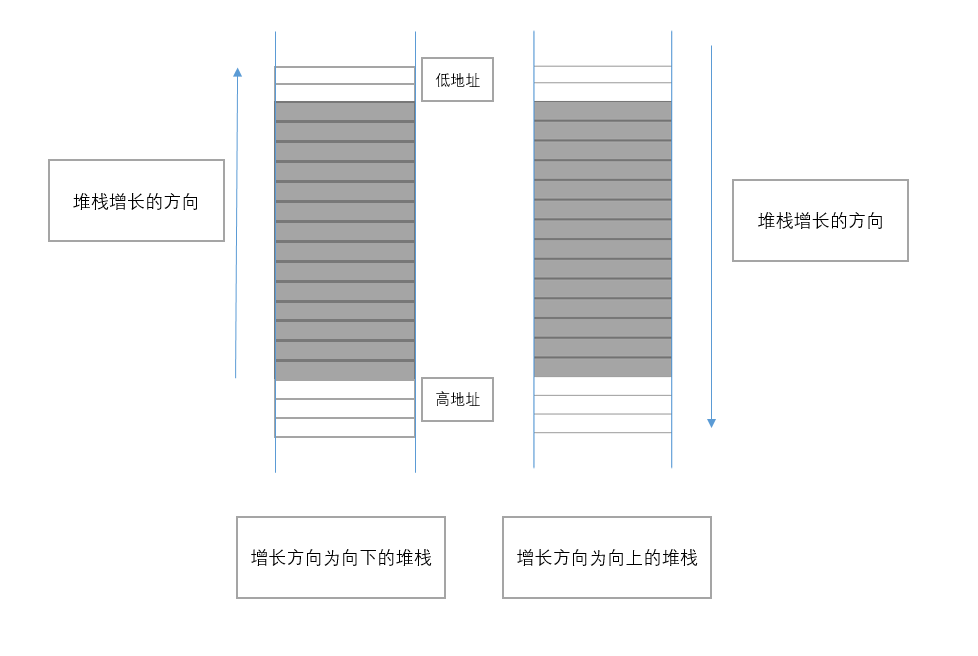




3）任务控制块链表



4）任务堆栈（不同CPU架构的栈在内存中的可能有不同的增长方向）



typedef struct {

void \*pData;

INT16U Opt;

void (\*Task)(void\*);

void \* Handle;

INT32U Id;

INT32 Exit;

} OS\_EMU\_STK;

3.2.3算法

**任务就绪表结构**：系统对于就绪表主要有三个操作：登记、注销和从就绪表的就绪任务中得知具有最高优先级任务的标识。

登记就是当某个任务处于就绪状态时，系统将该任务登记在任务就绪表中，即在就绪表中将该任务的对应位置置1.

在程序中，可用类似于下面的代码把优先级别为prio的任务置为就绪状态：

OSRdyGrp|= ptcb->OSTCBBitY;

OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

其中： ptcb->OSTCBBitY = (OS\_PRIO)(1uL << ptcb->OSTCBY);

ptcb->OSTCBBitX = (OS\_PRIO)(1uL << ptcb->OSTCBX);

ptcb->OSTCBY= (INT8U)(prio >> 3u);

ptcb->OSTCBX= (INT8U)(prio & 0x07u);

注销就是当某个任务需要脱离就绪状态时，系统在就绪表中将该任务的对应位置置0。对应伪代码如下：

If((OSRdyTblp[pri>>3]&=-OSMapTbl[prio&0x07])==0)

OSRdyGrp&=-OSMapTbl[prio>>3];

**最高优先级就绪任务的查找：**系统调度器总是把CPU控制权交给优先级最高的就绪任务，因此调度器就必须具有从任务就绪表中查找最高优先级任务的能力。EMOS调度器用于获取优先级别最高的就绪任务的伪代码如下：

Y=OSUnMapTbl[OSRdyGrp]; //获得优先级别的D5、D4、D3位

X=OSUnMapTbl[OSRdyTbl[y]]; //获得优先级别的D2、D1、D0位

Prio=(y<<3)+y; //获得就绪任务书的优先级

代码执行后，找的最高优先级就绪任务书的优先级别（即任务的标识）。其中OSUMapTbl[]同样是EMOS为提高查找速度定义的一个数组，共有什么256个元素（16X16）：

INT8U const OSUnMapTbl[256] = {

0u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x00 to 0x0F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x10 to 0x1F \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x20 to 0x2F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x30 to 0x3F \*/

6u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x40 to 0x4F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x50 to 0x5F \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x60 to 0x6F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x70 to 0x7F \*/

7u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x80 to 0x8F \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0x90 to 0x9F \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xA0 to 0xAF \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xB0 to 0xBF \*/

6u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xC0 to 0xCF \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xD0 to 0xDF \*/

5u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, /\* 0xE0 to 0xEF \*/

4u, 0u, 1u, 0u, 2u, 0u, 1u, 0u, 3u, 0u, 1u, 0u, 2u, 0u, 1u, 0u /\* 0xF0 to 0xFF \*/

};

找到最高优先级任务与当前任务作替换后，再调用任务级调度器交换CPU使用权就可以完成任务切换了。

**3.3 中断和时间管理**

3.3.1功能

**中断：**

在应用程序中经常有一些代码段必须不受任何干扰地连续运行，这样的代码段叫做临界段。因此，为了使临界段在运行时不受中断所打断，在临界段代码前必须用关中断指令使CPU屏蔽中断请求，而在临界段代码后必须用开中断指令解除屏蔽使得CPU可以响应中断请求 由于各厂商生产的CPU和C编译器的关中断和开中断的方法和指令不尽相同，为增强EMOS的可移植性（即在EMOS函数中尽可能地不出现汇编语言代码），EMOS用两个宏来实现中断的开放和关闭，而把与系统的硬件相关的关中断和开中断的指令分别封装在这两个宏中：

OS\_ENTER\_CRITICAL( )

OS\_EXIT\_CRITICAL( )

在中断服务程序中调用的负责任务切换工作的函数OSIntCtxSw( )叫做中断级任务切换函数。

**时间管理：**

1）提供的延时函数

由于嵌入式系统的任务是一个无限循环，并且μC/OS-II还是一个抢占式内核，所以为了使高优先级别的任务不至于独占CPU，可以给其他任务优先级别较低的任务获得CPU使用权的机会，μC/OS-II规定：除了空闲任务之外的所有任务必须在任务中合适的位置调用系统提供的函数OSTimeDly( )，使当前任务的运行延时（暂停）一段时间并进行一次任务调度，以让出CPU的使用权。

其它时间管理功能有：

2）取消任务延时函数

3）设置系统时间函数

4）获得系统时间函数

3.3.2数据结构

此部分无自己的数据结构，均是在实时内核的基础上实现的功能，在调用中所用到的数据结构见第一部分的实时内核中。

3.3.3算法

系统响应中断的过程为：系统接收到中断请求后，这时如果CPU处于中断允许状态（即中断是开放的），系统就会中止正在运行的当前任务，而按照中断向量的指向转而去运行中断服务子程序；当中断服务子程序的运行结束后，系统将会根据情况返回到被中止的任务继续运行或者转向运行另一个具有更高优先级别的就绪任务。

服务子程序运行结束之后，系统将会根据情况进行一次任务调度去运行优先级别最高的就绪任务，而并不是一定要接续运行被中断的任务的。

系统时钟中断服务函数执行过程如下：

保存CPU寄存器；

调用OSIntEnter( ); //记录中断嵌套层数

if (OSIntNesting = = 1;

{

OSTCBCur->OSTCBStkPtr = SP; //保存堆栈指针

}

调用OSTimeTick( ); //节拍处理

清除中断；

开中断；

调用OSIntExit( ); //中断嵌套层数减一

恢复CPU寄存器；

中断返回；

数OSTimeTick( )的任务，就是在每个时钟节拍了解每个任务的延时状态，使其中已经到了延时时限的非挂起任务进入就绪状态。

**3.4 内存管理**

3.4.1功能

应用程序在运行中为了某种特殊需要，经常需要临时获得一些内存空间，因此作为一个比较完善的操作系统必须具有动态分配内存的能力。能否合理、有效地对内存储器进行分配和管理，是衡量一个操作系统品质的指标之一。特别地对于实时操作系统来说，还应该保证系统在动态分配内存时，它的执行时间必须是可确定的。

EMOS改进了ANSI C用来动态分配和释放内存的malloc( )和free( )函数，使它们可以对大小固定的内存块进行操作，从而使malloc( )和free( )函数的执行时间成为可确定的，满足了实时操作系统的要求。EMOS的内存管理目前主要功能是对内存进行简单管理，并向系统内核作底层支撑，当系统内核启动时，需要为任务分配加载内存。也可以用于应用程序的动态分配和释放内存。

3.4.2数据结构

**内存控制块**的结构如下：

typedef struct {

void \*OSMemAddr; //内存分区的指针

void \*OSMemFreeList; //内存控制块链表的指针

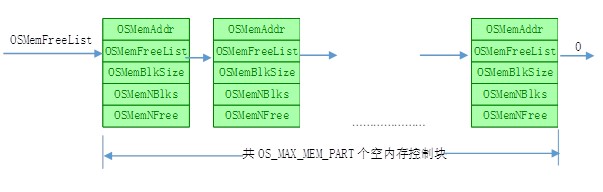
INT32U OSMemBlkSize; //内存块的长度

INT32U OSMemNBlks; //分区内内存块的数目

INT32U OSMemNFree; //分区内当前可分配的内存块的数目

} OS\_MEM;

**内存控制块链表：**



3.4.3算法

EMOS对内存进行两级管理，即把一个大片连续的内存空间分成了若干个分区，每个分区又分成了若干个大小相等的内存块来进行管理(验收时只实现了用简单的分块思路，经老师提醒后，这一周又加上了分区管理，变成了两级管理)。操作系统以分区为单位来管理动态内存，而任务以内存块为单位来获得和释放动态内存。内存分区及内存块的使用情况则由表——内存控制块来记录。

应用程序如果要使用动态内存的话，则要首先在内存中划分出可以进行动态分配的区域，这个划分出来区域叫做内存分区，每个分区要包含若干个内存块。EMOS要求同一个分区中的内存块的字节数必须相等，而且每个分区与该分区的内存块的数据类型必须相同。

为了使系统能够感知和有效地管理内存分区，EMOS给每个内存分区定义了一个叫做内存控制块（OS\_MEM）的数据结构。系统就用这个内存控制块来记录和跟踪每一个内存分区的状态。当应用程序调用函数OSMemCreate( )建立了一个内存分区之后，内存控制块与内存分区和内存块之间的关系如图

在内存中划分一个内存分区与内存块的方法非常简单，只要定义一个二维数组就可以了，其中的每个一维数组就是一个内存块。例如，定义一个用来存储INT16U类型数据，有10个内存块，每个内存块长度为10的内存分区的代码如下：

INT16U IntMemBuf[10][10];

上面这个定义只是在内存中划分出了分区及内存块的区域，还不是一个真正的可以动态分配的内存区，只有当把内存控制块与分区关联起来之后，系统内核才能对其进行相应的管理和控制，它才能是一个真正的动态内存区。

**3.5任务管理**

3.5.1功能、

任务管理主要是对系统中的所有任务（包括已被删除但TCB仍在空链表中的任务）其主要功能如下：

创建任务；

删除任务；

挂起任务；

唤醒任务；

3.5.2数据结构

此部分没有自己的数据结构，均是在实时内核的基础上实现的任务管理功能，在调用中所用到的数据结构见第一部分的实时内核中。

3.5.3算法

应用程序在创建一个新任务的时候，必须把在系统启动这个任务时CPU各寄存器所需要的初始数据（任务指针、任务堆栈指针、程序状态字等等），事先存放在任务的堆栈中，任务堆栈的初始化就是对该任务的虚拟处理器的初始化（复位）。在应用程序中定义任务堆栈的栈区非常简单，即定义一个OS\_STK类型的一个数组并在创建一个任务时把这个数组的地址赋给该任务就可以了。

当进行系统初始化时，初始化函数会按用户提供的任务数为系统创建具有相应数量的任务控制块并把它们链接为一个链表。由于这些任务控制块还没有对应的任务，故这个链表叫做空任务块链表。即相当于是一些空白的身份证。

当应用程序调用函数OSTaskCreate( )创建一个任务时，这个函数会调用系统函数OSTCBInit ( )来为任务控制块进行初始化。这个函数首先为被创建任务从空任务控制块链表获取一个任务控制块，然后用任务的属性对任务控制块各个成员进行赋值，最后再把这个任务控制块链入到任务控制块链表的头部。

以下是任务转换图：（包括各个任务状态发生切换时所触发的函数动作）



加上任务挂起与任务唤醒后的状态转换关系：



**3.6 CPU移植相关**

3.6.1功能

此部分主要是处理与处理器架构相关的实时内核代码，包括关于X86架构栈初始化，关中断，开中断的宏实现，以及利用WINDOWS API实现的线程间CPU切换功能。切换功能分为两部分，一种为任务级的切换，一种为中断级的切换。

3.6.2数据结构

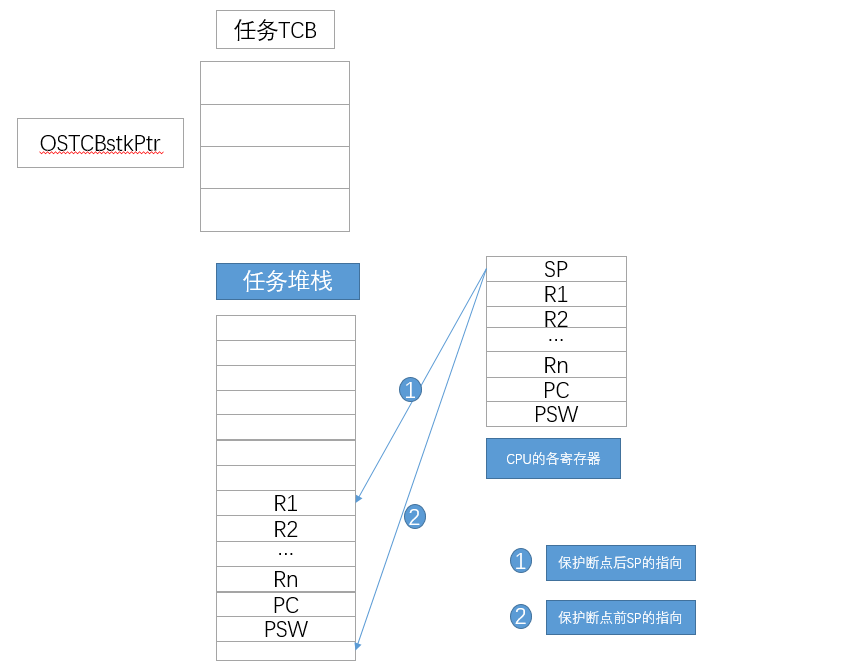
此部分也没有特别的数据结构，栈相关的数据结构均参照第一部分。

3.6.3算法

由于系统存在着多个任务，于是系统如何来识别并管理一个任务就是一个需要解决的问题。识别一个任务的最直接的办法是为每一个任务起一个名称。

由于EMOS中的任务都有一个惟一的优先级别，因此EMOS是用任务的优先级来作为任务的标识的。所以，任务控制块还要来保存该任务的优先级别。

一个任务的任务控制块的主要作用就是保存该任务的虚拟处理器的堆栈指针寄存器SP。具体如下图所示：

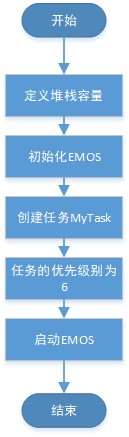


而任务的切换就是将当前的CPU中的SP指针保存到自己的TCB中，同时将通用寄存器值保存到TCB中相应的位置，这个过程就叫任务的现场保护，接下来再通过第一部分讲到的寻找就绪表中最高优先级任务的算法得到要切换所任务TCB,将TCB中的SP值搬运到CPU中，同时TCB中保存的通用寄存器的值也复制到CPU中，这个过程就叫现场恢复，整个任务的切换过程，就是完成了任务的上下文与CPU寄存器值的相互切换工作。

**4 程序设计与实现**

**4.1 程序流程图**

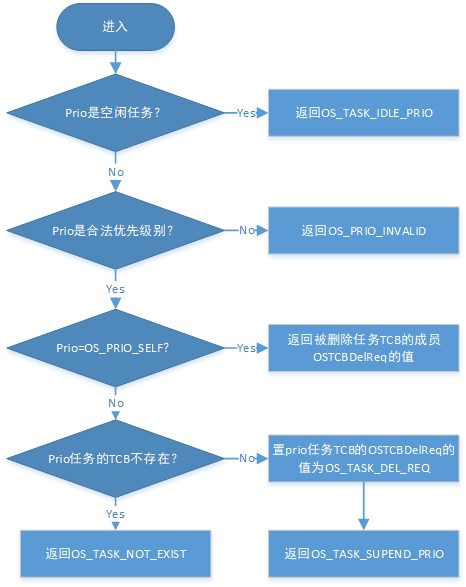
1）实时内核启动流程：



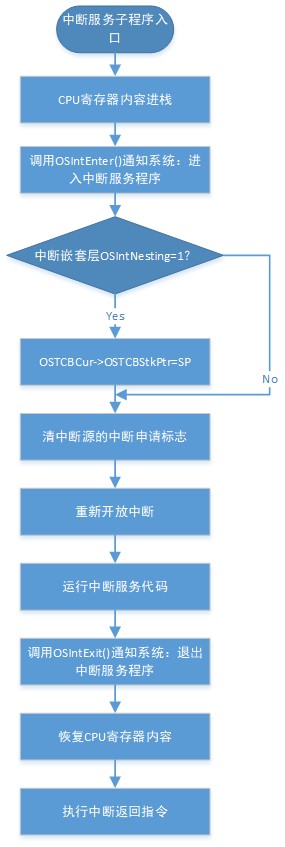
2）创建新任务流程：



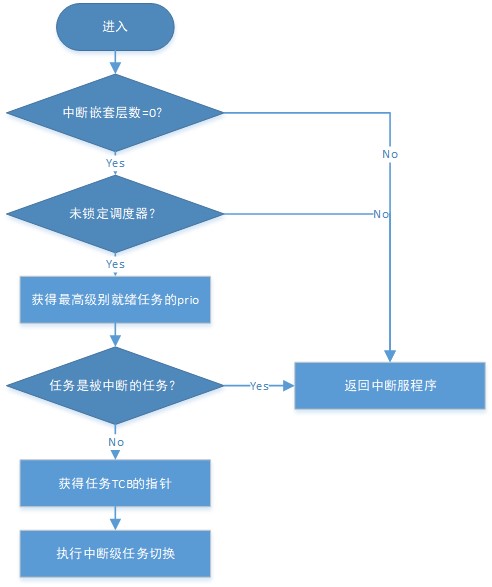
3）请求删除任务流程：



4）中断服务子程序流程图



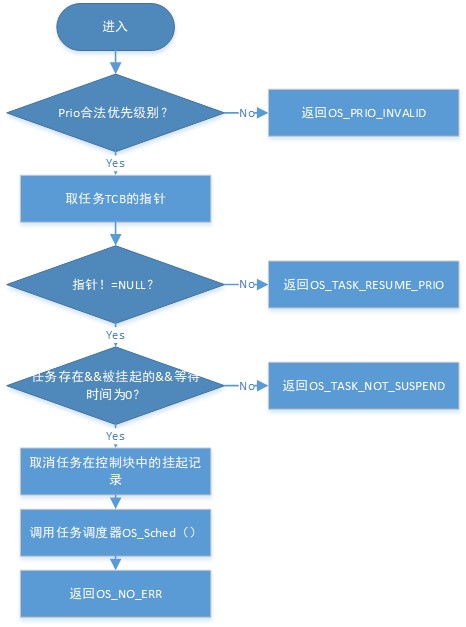
5）退出中断服务函数OSIntExit().函数流程图如下：



6）任务挂起流程图：



7）任务唤醒流程图：



**4.2 程序说明**

我们这个简单的实时操作系统内核起名为：“EMOS”

**开发平台**：Visual Studio 2017, Windows 10

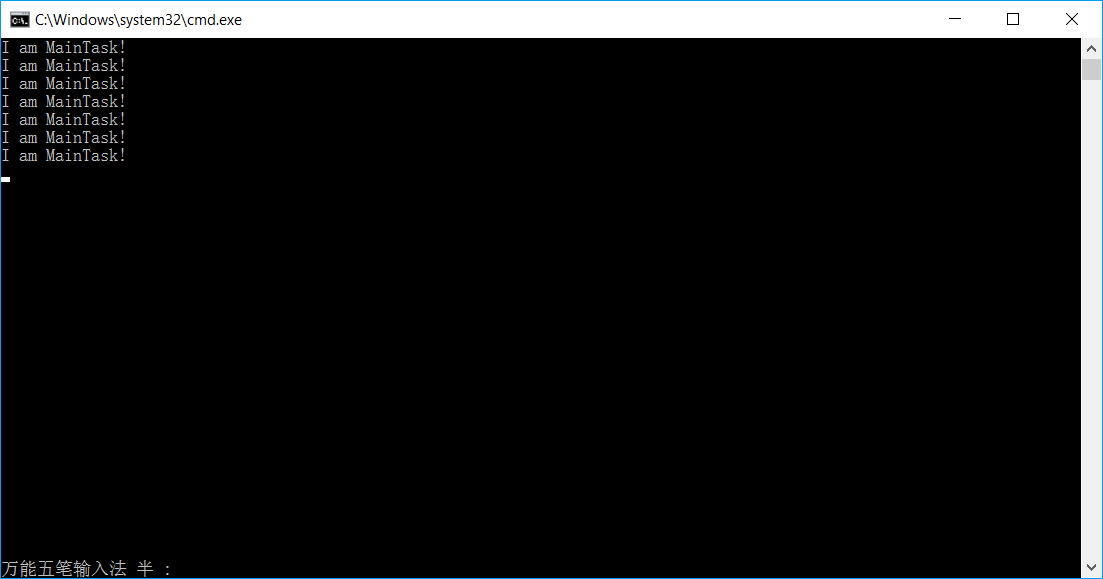
程序的结构很清晰，一共分为七个模块：

|  |  |  |
| --- | --- | --- |
| **模块名** | **文件名** | **内容描述** |
| **核心代码** | **os\_core.c** | **实时内核核心数据结构与算法的实现** |
| **emos.h** | **包含所有内核代码需要的数据结构定义和宏定义以及函数声明** |
| **任务管理** | **os\_task.c** | **创建、删除、挂起、唤醒任务等功能函数** |
| **时间管理** | **os\_time.c** | **时间管理功能函数** |
| **内存管理** | **os\_mem.c** | **内存管理功能函数** |
| **移植文件** | **os\_cpu\_c.c** | **平台移植相关代码** |
| **os\_cpu.h** | **平台移植相关头文件** |
| **内核配置** | **os\_cfg\_r.h** | **EMOS内核功能配置头文件** |
| **主函数** | **main.c** | **内核测试相关代码** |
| **main.h** | **内核测试相关头文件** |

**4.3 实验结果**

在测试程序过程中：

我们首先创建了一个主任务，优先级赋为5，堆栈大小为512bytes:

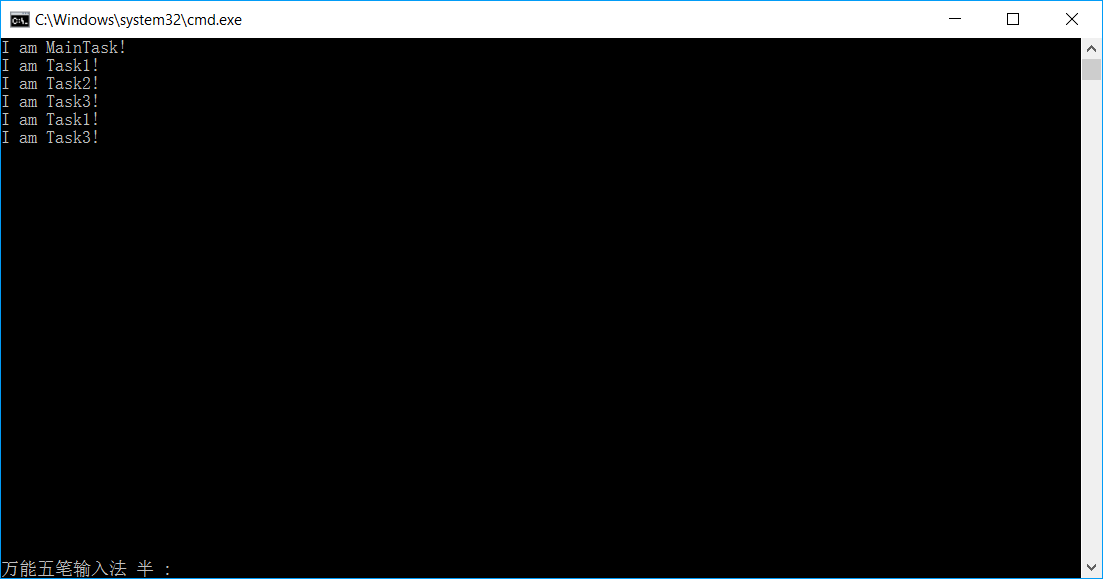
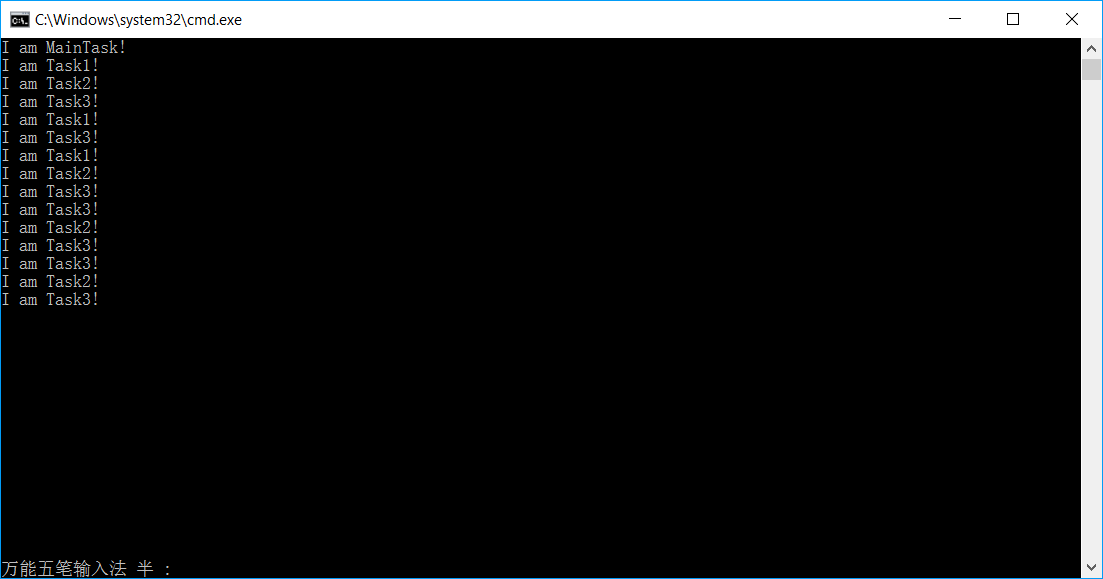
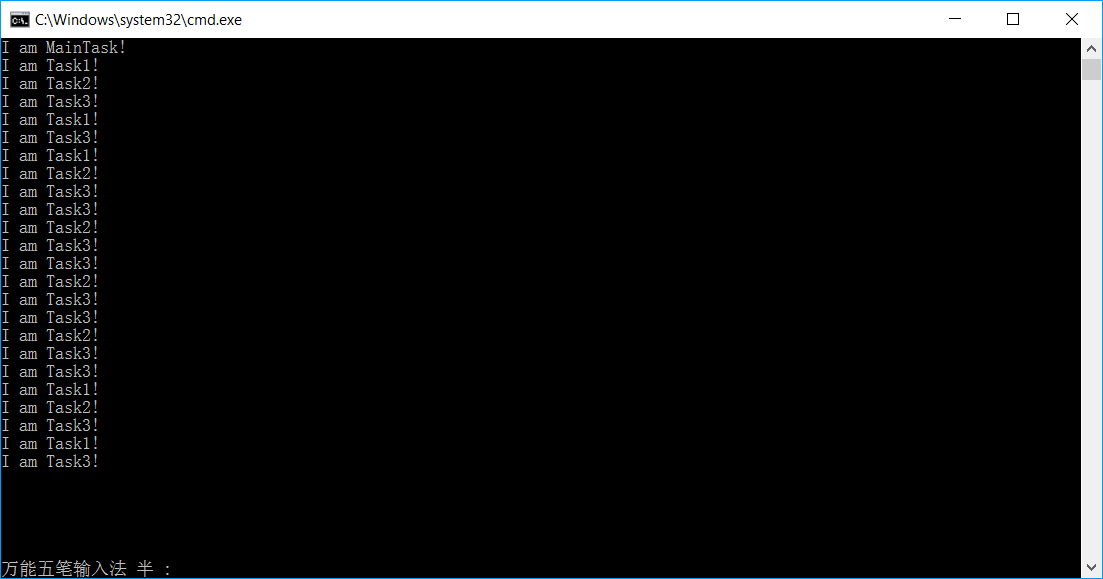
然后启动操作系统，调度主任务开始运行，运行周期设为1s:

然后在主任务中创建了三个用户任务，优先级分别为7、9、10

栈空间均为人512字节

它们的运行周期分别为什么1s、2s、1s

创建完用户任务后，主任使命完成，所以删除自身。

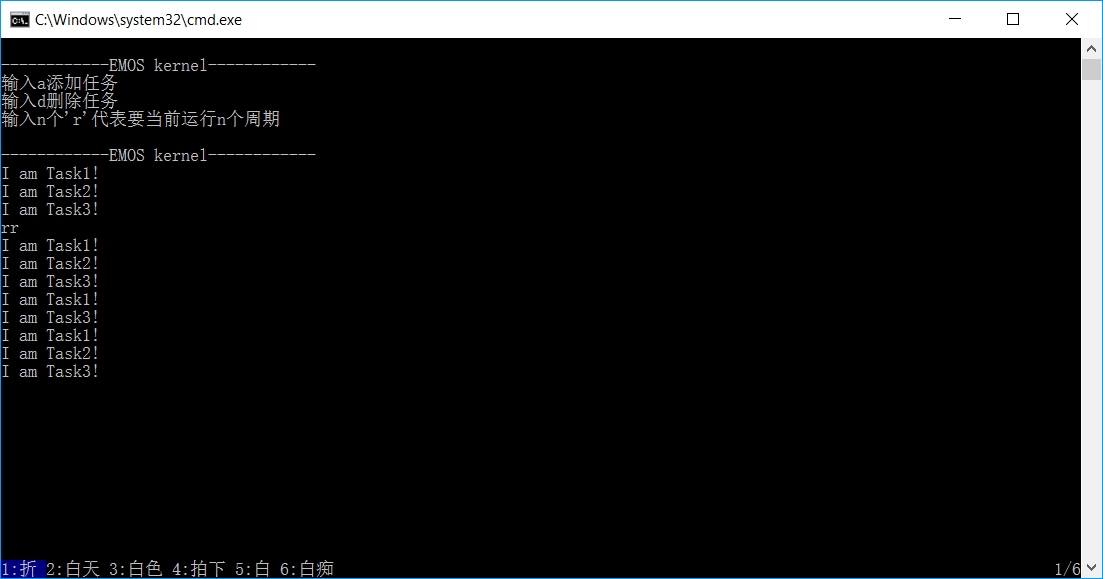
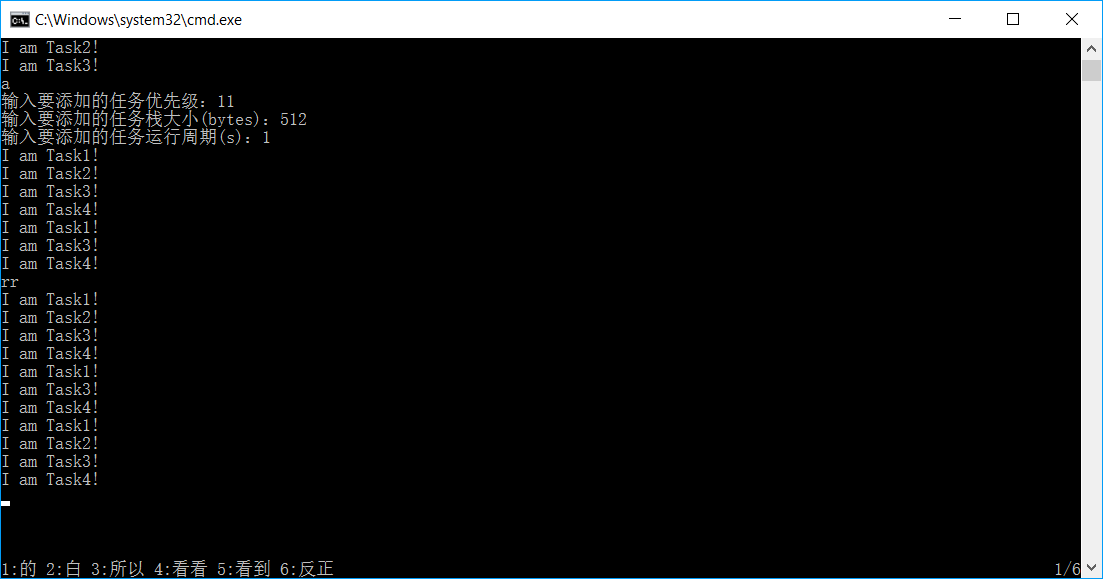
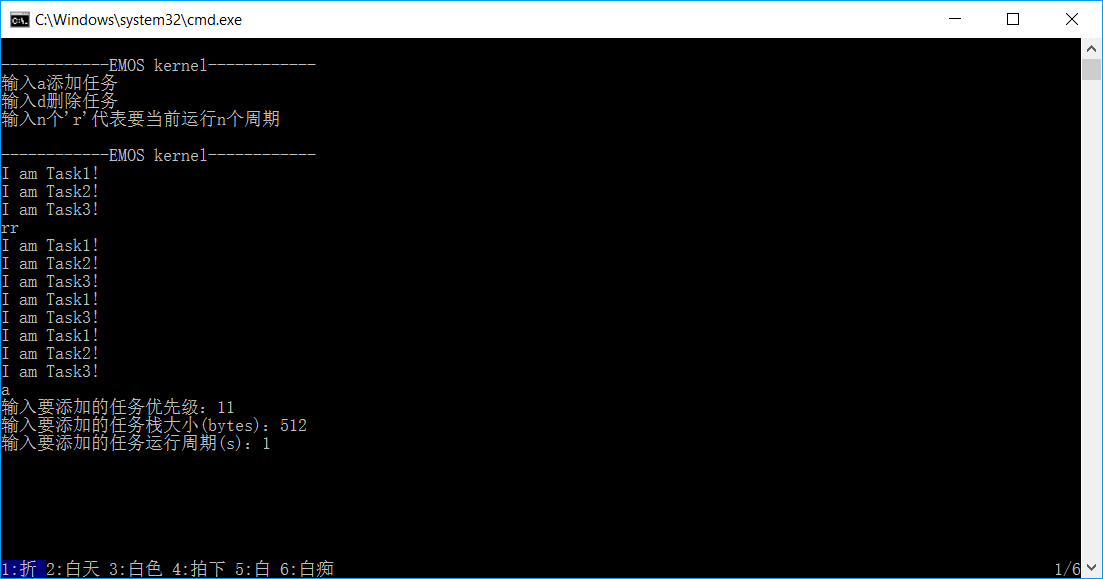
每个任务中执行的内容分别是打印出自己的任务名称。当任务1运行完第一个周期后将自身挂起：在10s后，即任务2运行了解到第5个周期时，此时再唤醒任务1：

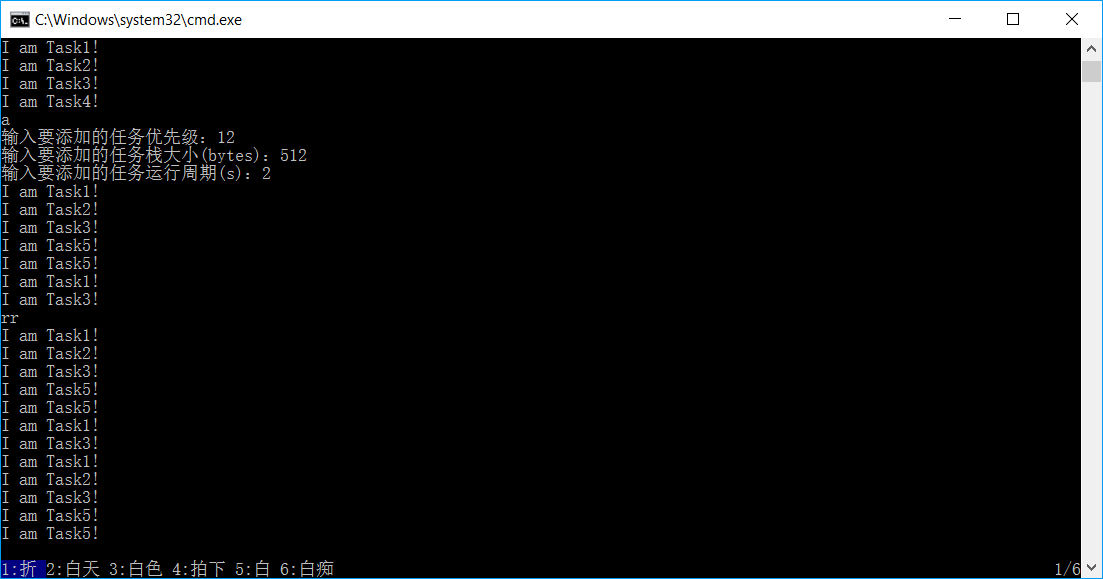
以上测试：包括了任务管理中的四部分功能，创建任务，删除任务，挂起任务，唤醒任务。

在任务管理的测试中也包括了对内存分配与释放的测试（创建任务会申请分配内存，删除任务会释放内存）也包括了对时间管理的测试（每个任务都定义了deadline和时钟周期，在运行时严格按周期计算，准确无误）。

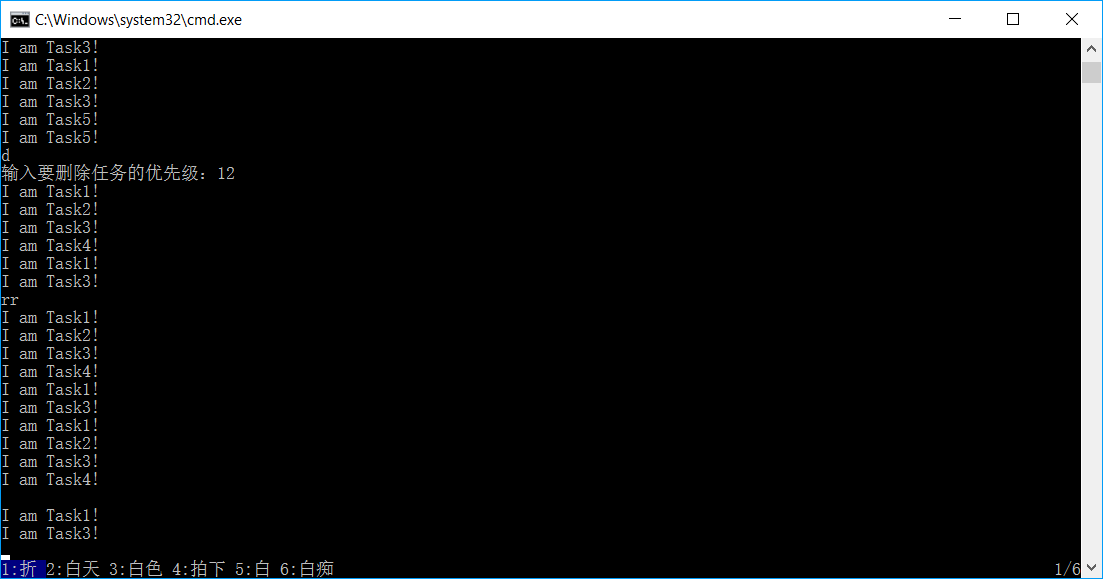
**动态任务管理接口：**

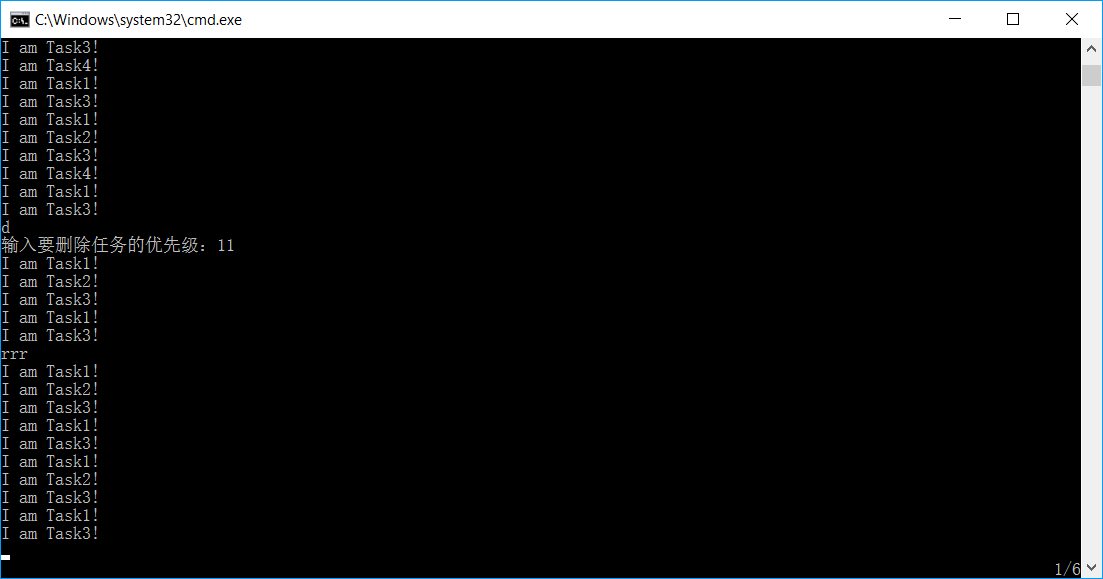
在老师的建议下增加了用户在系统运行过程中动态增删任务的需求：

当系统运行时通过输入n个‘r’代表要让现有任务运行n个周期：当输入‘a’时代表用户增加一个任务：



当输入‘d’时删除一个任务：





**5 结论**

本次课程设计，我们采用了C语言编写完成了一个简单的实时操作系统内核，通过测试，这个简单的操作系统可以用于实时系统的任务管理，内存分配，以及时间管理。在移植性上也给实时内核的用户提供了最大便利。

当然，由于时间有限，这个简单的实时内核也存在着许多需要完善的地方，比如缺少任务间的通信模块，没有对常见的CPU架构进行移植示例等。所以这次课设的结束并不意味着EMOS的开发就结束了，我还想把这个小内核继续完善，并移植到ARM架构上，满足自己在ARM上的应用程序提供实时任务调度的需求。

**6 参考文献**

1、周苏等编著.《*操作系统原理实验*》.北京：科学出版社.2003.

2、任哲等编著.《嵌入式实时操作系统μCOS-II原理及应用》.北京：北京航空航天大学出版社,2009.

3、徐虹等编著.*《操作系统实验指导——基于Linux内核》*. 北京：国清华大学出版社.2004.

4、陈向群等编著.《Windows内核实验教程》.北京：机械工业出版社.2002

**7 收获、体会和建议**

**唐在洋：**

我觉得这次最大的收获就是把在书本上学到的一些操作系统原理，包括之前在实验室看的一些实时操作系统的原理与算法，都从具体实现的角度再学习了一遍，这和仅学习原理和算法是不一样的。这就相当于纸上谈兵与实际演习的关系一样吧，之前学习理论知识大概可以说是纸上谈兵吧，可能要给别人讲清楚是什么，或者在考场上答题这是完全没有问题的，但一旦要将这些原理和算法运用起来，你才会发现，有些地方按照你原来的理解是完全行不通的。

每一次课程设计都让我学到了在平时课堂不可能学到的东西。所以我对每一次课程设计的机会都非常珍惜。不一定我的课程设计能够完成得有多么完美，但是我总是很投入的去研究去学习。所以在这两周的课设中，熬了2个通宵，生物钟也严重错乱了，但是每完成一个任务我都兴奋不已。一开始任务是任务，到后面任务就成了自己的作品了。[总结](http://www.liuxue86.com/fanwen/zongjie/)一下有以下体会。

　　1、网络真的很强大，用在学习上将是一个非常高效的助手。几乎所有的资料都能够在网上找到。也因为这样，整个课程设计下来，我浏览的相关网页已经超过了100个(不完全统计)。当然网上的东西很乱很杂，自己要能够学会筛选。

　　2、同学间的讨论，这是很重要的。老师毕竟比较忙。对于课程设计最大的讨论伴侣应该是同学了。能和学长学姐讨论当然再好不过了，没有这个机会的话，和自己班上同学讨论也是能够受益匪浅的。大家都在研究同样的问题，讨论起来，更能够把思路理清楚，相互帮助，可以大大提高效率。

　　3、敢于攻坚，越是难的问题，越是要有挑战的[心理](http://www.liuxue86.com/xinli/" \t "_blank)。这样就能够达到废寝忘食的境界。当然这也是不提倡熬夜的，毕竟有了精力才能够打持久战。但是做课设一定要有状态，能够在吃饭，睡觉，上厕所都想着要解决的问题，这样你不成功都难。

　　4、最好在做课设的过程中能够有记录的习惯，这样在写实验报告时能够比较完整的回忆起中间遇到的各种问题。比如当时我遇到我以前从未遇到的段错误的问题，让我都不知道从何下手。在经过大量的资料查阅之后，我对段错误有了一定的了解，并且能够用相应的办法来解决。

　　在编程中以下几类做法容易导致段错误,基本是是错误地使用指针引起的

　　1)访问系统数据区，尤其是往系统保护的内存地址写数据，最常见就是给一个指针以0地址

　　2)内存越界(数组越界，变量类型不一致等) 访问到不属于你的内存区域

　　3)其他

　　例如：

　　<1>定义了指针后记得初始化，在使用的时候记得判断是否为null

　　<2>在使用数组的时候是否被初始化，数组下标是否越界，数组元素是否存在等

　　<3>在变量处理的时候变量的格式控制是否合理等

致谢：首先感谢林老师在本学期的操作系统课上的悉心指导与付出。在本次课设中虽然没有多少问题麻烦到老师，但我能感受到老师的认真负责是在以往课设的指导老师中很少见到的。在验收过程中老师提出了保贵意见，例如，在内存管理方式上老师提出在分区管理上也可以加一层分区管理并不影响实时系统的时间确定性，我们觉得很有道理，之前用的方法确实过于简单，可能不满足某些系统用户的需求。所以我们就照老师的修改了算法。再次致谢林老师。

**黄世华**：

通过本次课程设计，对操作系统课程知识有了更加巩固性的理解，对嵌入式实时操作系统有了更深入的了解。本次课程设计中，我们要做的是一个微型实时性操作系统，有着任务创建和被调度，优先级的排序，内存的分配，处理器实现调度和处理任务等工作。本次课程设计中我主要负责的是CPU处理器模块，很多工作都是基于调用现有windows操作系统上的API接口函数，完成处理器对任务的调度和处理。非常感谢本次课程设计的所有组员，和他们一起学习真的很开心，收获很大，特别感谢唐在洋同学，本次课程设计中很多部分都是他带领我们学习的，没有他，我们也不可能轻轻松松完成本次课程设计，最后特别感谢林老师对本次课程设计中的各种点评，认识到自己的不足后，在验收结束后继续完善了对本次课程设计的。通过本次课程设计，对操作系统课程知识有了更加巩固性的理解，对嵌入式实时操作系统有了更深入的了解。本次课程设计中，我们要做的是一个微型实时性操作系统，有着任务创建和被调度，优先级的排序，内存的分配，处理器实现调度和处理任务等工作。本次课程设计中我主要负责的是CPU处理器模块，很多工作都是基于调用现有windows操作系统上的API接口函数，完成处理器对任务的调度和处理。非常感谢本次课程设计的所有组员，和他们一起学习真的很开心，收获很大，特别感谢唐在洋同学，本次课程设计中很多部分都是他带领我们学习的，没有他，我们也不可能轻轻松松完成本次课程设计，最后特别感谢林老师对本次课程设计中的各种点评，认识到自己的不足后，在验收结束后继续完善了对本次课程设计的。

**陈治铭**：

课程设计进行了两周，这是一次将书本上的知识应用到实践的考验。经过大概一周的设计，先是参透书本上的知识，并在网上搜集相关资料，自学了一些与 uC/OS-II任务调度，内存管理方面的知识。

在这次课程设计中，我通过查阅网上的资料，了解了实时操作系统内存管理的各种算法，通过阅读现有的操作系统的内存管理方面的代码，我逐渐熟悉了这方面的设计方法，在这过程中，提高了我的资料检索能力与阅读代码能力，我的部分是内存管理，在网上及书本中已经有了很多关于内存管理的算法，最终我选择了最适合实时系统的算法，虽然算法理解起来并不难，但是编程实现确是不易，在编程实现过程中，我遇到了很多困难，在组长唐在洋的带领下，我克服了这些困难，同时也使自己的编程能力和项目设计能力得到了提高。在这次课程设计之后，我收获了宝贵的经验，这将在以后的工作学习生活中起到非常大的作用。这次课程设计将是这四学年最后的一次课程设计，在这，我向一同陪伴我们走过的老师们表示衷心的感谢！

**王清弘**：

本次操作系统课设，我们做的实时类操作系统，我分配到的任务是任务管理。任务管理是操作系统课上学习的一个重要内容，涉及到了创建，删除，挂起和唤醒。这四个任务调度是我在书本上了解的，也明白了具体的定义，但是对于代码段的设计方法一直不太清楚。而课设提供给我们的正是这样的机会，去了解明白和掌握如何实施这些功能。还记得期末考试前的复习，我看到书本上的这些内容，虽然这部分的内容并不是很难，我也一直想亲手实践一下，以便于我更好的理解，但是却因为更方面的原因没有这么做，而这次的课设正是提供了这样的机会，让我感受颇深。还要说的是，关于实时的实现，让我更对不同的算法对于设计的简便的益处有了更深入的了解。比如说系统需要对不同优先级拥有相同时间的的调度，而这种算法的实现是通过位视图。通过8乘8的矩阵找寻优先级的任务，以及一个12乘12的矩阵就能完成对任务调度的相同时间复杂度，这样就能实现实时操作系统的调度。一开始接触这种算法的时候我其实是不太能够理解的，也请问了很多的同学，最终才明白了其中的含义。其实这种实时性的方法可以应用到许多的领域，比如数据库的构建，嵌入式的应用。我们这次的设计其实是一种基于嵌入式的操作系统的构建，也让我学习到了许多关于嵌入式方面的知识，我们的组长对于嵌入式了解的比较多，通过与他一起做这次的设计，我明白了实时性对于嵌入式的重要性，因为实时操作系统的正确性不仅依赖系统计算的逻辑结果，还依赖于产生这个结果的时间，而在嵌入式对于结果的正确性拥有更高的要求，嵌入式要求同步的要求是非常高的。总的来说，这次操作系统的课设不仅让我对操作系统具体代码段的设计有了更深入的了解，而且让我对于嵌入式的设计有了一定的接触，这对于我是非常有好处的。