**嵌入式实时操作系统**

**实验报告一**

**姓名： 蒋叶琪**

**班级： 软件2班**

**学号： S201625035**

### 一、实验目的

任务管理作为内核的核心部分，具有任务调度、创建任务、删除任务、挂起任务、解挂任务等功能。uCOS -Ⅱ中对任务是精华之处，通过对任务进行管理，完成了任务并行的思想，同时根据遵循操作系统设计原则，uCOS -Ⅱ是以任务控制块的数据结构作为该嵌入式操作系统的统一的数据结构的。

本次试验通过改写OSTaskCreate ()、OSTaskDel()、OSTaskSuspend()、OstaskResume()等函数，掌握任务的管理方法，近而掌握uCOS -Ⅱ的编写精髓。

### 二、实验内容

**（1）Int OSTaskCreate ()**

{

1. 判定i是否有效。
2. 创建2个链表，一个叫freeTCBlist 6个长，一个叫currentTCBlist

2个长度。

1. 把freeTCBlist 链头摘下来加到currentTCBlist链头。

TCBarrary[i] 指向当前currentTCBlist链头。

④ 注册生成表 TaskReadytable[i>>3] 优先级i的位置为1。

⑤ 调用调度函数 OSschedule()。

}

**（2）Int OSTaskDel()**

{

1. 判定i是否有效。
2. 判定i=self 那么要调度。
3. 清注册表。
4. 删除链表：把currentTCBlist链头摘 下来加到 freeTCBlist链头, TCBarrary[i]指向null。

}

**（3）Int OSTaskSuspend()**

{

1. 判定挂起是否自己（SELF，i），如果挂起自己，则调度。
2. 清就绪表。
3. OSTCBprioTbl[i].OsTCBStat=SUSPEND。

}

**（4）Int OstaskResume()**

{

1. 判定I OSTCBprioTbl[i].OsTCBStat=SUSPEND。
2. 置位就绪表。
3. 调度。

}

### 三、实验原理及程序结构

（1）任务建立的函数主要建立该任务的堆栈和任务控制块的初始化。任务控制块（Task Control Block）TCB是存储任务信息的一张表，在uCOS中是一个结构体定义的，所有的任务的控制块构成一个一个双向链表。在之前需要判断设定的优先级的有效性，包括优先级是否越界和优先级是否已被占用。创建任务函数（OSTaskCreate）的代码如下：

#include "stdafx.h"

#include<stdlib.h>

#define NULL 0

typedef struct DuLNode //定义结点

{

struct DuLNode \*next;

struct DuLNode \*prior;

int data;

} DuLNode;

void Create\_DList(DuLNode\*head,int num) //创建双向链表

{

DuLNode\*p1,\*p2;

p1=head=(struct DuLNode\*)malloc(sizeof(struct DuLNode));

head->next=NULL; //头节点指针域置NULL

int i;

for(i=1;i<num;i++)

{

p2=(struct DuLNode\*)malloc(sizeof(struct DuLNode));

p1->next=p2;

p2->prior=p1; //新节点p2插入p1后，作为表尾

p1=p1->next; //p1向后移一位，方便后面结点插入

num--;

return head;

}

}

int OSTaskCreate(int i) // 创建任务函数

{

char TaskReadytable[8]={0,0,0,0,0,0,0,0}; //定义优先级就续表

LNode \*TCBarrary[8]={NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL}; //定义优先级就绪组

char OSMapTbl[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80}; //定义优先级映射表

TaskReadytable[i>>3]|=OSMapTbl[i&0x07]; //任务进入就绪

DuLNode \*freeTCBlist=NULL;

DuLNode \*currentTCBlist=NULL;

Create\_ DList (freeTCBlist,6);

Create\_ DList (currentTCBlist,2); //创建freeTCBlist、currentTCBlist链表

if (i<0||i>8||TCBarrary[i] != NULL||TaskReadytable[i>>3]==1) //判断当前是否越界或者占用。

return 1;// i不合格的情况下返回1

DuLNode \*p3;

DuLNode=(struct DuLNode \*)malloc(sizeof(struct DuLNode));

p3=freeTCBlist->next;

freeTCBlist->next=freeTCBlist->next->next;

currentTCBlist->next=p3;

p3->prior= currentTCBlist;

// 把freeTCBlist 链头摘下来加到currentTCBlist链头。

TCBarrary[i]=p3; // TCBarrary[i] 指向当前currentTCBlist链头。

TaskReadytable[i>>3]=1; //注册生成表 TaskReadytable[i>>3] 优先级i的位置为1。

OSschedule();

return 0;

}

（2）删除任务的函数会先检查删除任务的条件是否成立，如果不成立，则返回相应的错误码。一旦条件满足，TCB就会从任务链中移除，放到空闲任务表中。其中包括把本任务从就续表TaskReadytable[ ]中删除，把本任务从索引数组TCBarrary[ ]中删除，启动调度器运行下一个优先级最高的就绪任务。OSTaksDel()函数代码大体如下：

int OSTaskDel(int i)

{

OSTaskCreate(2)

LNode\*p4;

int self=6;

if(i<0 || i>8) //判断i是否有效，无效返回1

return 1;

else if (TaskReadytable[i>>3]==self) //i=self的话，启用调度函数

OSschedule(6);

else

{

TaskReadytable[i>>3]=0; //清空就续表

p4=(struct LNode\*)malloc(sizeof(struct LNode));

p4=currentTCBlist->next;

freeTCBlist->next=p4; //将任务放到空闲任务表中

TCBarrary[i]->next=NULL; //本任务从索引数组TCBarrary[ ]中删除

}

return 0;

}

（3）使用OSTaskSuspend( )可以讲一个任务挂起，该函数的参数是任务的优先级。通过赋予不用的参数可以挂起其他任务，也可以挂起自己。OSTaskSuspend挂起指定任务，知道通过唤醒任务对任务进行解挂。一个任务将自己挂起后，会引起任务的调度，实时内核将选取另外一个合适的任务执行。任务挂起的代码大体如下：

int OSTaskSuspend(int i)

{

OSTaskCreate(2)

int self=6;

if (i==self)

OSschedule(6);//如果挂起自己，调度

TaskReadytable[i>>3]=0; //清空就续表

TCBarrary[i]->next=NULL;

OSTCBprioTbl[i].OsTCBStat=SUSPEND；

return 0;

}

(4)任务唤醒要检查一下任务是否被挂起，然后把任务状态从挂起改为就绪。同时还要检查有没有因为延时而挂起，如果是延时挂起，则不能唤醒。OstaskResume函数的代码大体如下：

void OSTaskResume(\*prio)

{

DuLNode \*tcb

OSTaskCreate(2)

int self=6;

if (OSTCBprioTbl[i].OsTCBStat=self)

TaskReadytable[i>>3]=1; //置位就续表

return 0;

}

### 四、 总结

通过本次试验，我更加熟悉了任务生成、任务删除、任务挂起等任务的工作流程，对操作系统的工作原理有了更加深入的认识。在这个过程中，不仅巩固了数据结构的基础知识，更是对uCOS源码的深入学习和挖掘。在本次试验中，同时也暴露了自己许多问题，例如基础知识掌握不牢、对源码的理解力尚待提高，这些问题的暴露也提醒了自己戒骄戒躁，不断努力，时刻保良好的学习态度。

**嵌入式实时操作系统**

**实验报告二**

**姓名： 蒋叶琪**

**班级： 软件2班**

**学号： S201625035**

### 一、 实验目的

资源管理是操作系统都要考录的一个重要部分，有的资源可以同时使用（资源共享），有的资源只能轮流使用（资源互斥），最后任务呈现出一个先后顺序（任务同步）。为了能够表示资源共享和互斥、任务同步，信号量被用到嵌入式操作系统中来。信号量主要有两类操作，得到信号量操作（pend操作）和释放信号量操作（post操作）。本次试验通过改写OSSemCreate()、OSSemPend()、OSSemPost()函数，进一步了解时间控制块和资源管理机制。

### 二、 实验内容

1. **（1）OSSemCreate()**

{

1. 从ECB摘链表头
2. 赋值

}

**（2） OSSemPend()**

{

1. 判定计数是否小于0，如果小于0

{

则等待状态设成等信号量

让当前的任务从就绪队列里出来

调度

}

1. 当前ECB计数-1。

}

**（3）OSSemPost()**

{

1. 判定当前ECB里有没有等的任务，如果有

{

获取ECB 事件表里最高任务优先级

让该任务回就绪表

调度

}

②没有等的，则让ECB的计数+1

}

### 三、 实验原理及程序结构

（1）在使用信号量之前，要调用OSSemCreate()来创建一个信号量。信号量的作用有：允许一个任务和其他任务或者中断同步，取得设备的使用权，标志事件的发生。该函数只有OSEventCnt唯一的参数，用于初始化信号量的计数器初始值。创建信号量函数将返回一个指针\*OSEventPtr。下面是OSSemCreate()的大体代码。

#include "stdafx.h"

#include<stdlib.h>

#define NULL 0

struct OSEvent //定义ESB基本数据结构

{

struct OSEvent \*next;

struct OSEvent \*prio;

int number;

int OSEventType; //定义事件类型

int OSEventGrp; //任务所在组

int OSEventCnt; //计数器（信号量）

void \*OSEventPtr; //指向消息或消息队列的指针

int OSEventTbl[8]; //等待任务列表，长度为8

};

OSEvent \*Create\_List(OSEvent \*head , int num) //创建链表函数

{

P2=head = (struct OSEvent \*)malloc(sizeof(OSEvent));

head->next = NULL;

p2->next = NULL;

fori(i=0,i<num,i++)

{

OSEvent \*p1= (struct OSEvent \*)malloc(sizeof(OSEvent));

P2->next=p1;

P1->prio=p2; //后插法插入结点

P1->OSEventType = 0;

n->OSEventGrp = 0;

n->OSEventCnt = 0;

for(int i=0;i<8;i++)

{

n->OSEventTbl[i]=0;

} // 使p1结点的任务所在组、事件类型、计数器等置为0

p2=p2->next;

}

return head;

}

int OSSemCreate (int i) // 创建信号量函数

{

Create\_ List (freeECBlist,6);

Create\_ List (currentECBlist,2); //创建空闲事件控制块链和等待事件控制块

链

OSEvent \*p2= (struct OSEvent \*)malloc(sizeof(OSEvent));

P2= freeECBlist->next;

freeECBlist->next= freeECBlist->next->next;//从空闲事件控制块链中取得一

个ECB

p2->OSEventType = 1;

p2->OSEventCnt = 1; //赋值

currentECBlist->next=NULL; //初始化等待事件链表

return 1;

}

（2当任务调用OSSemPend()函数时，如果信号量的值大于0，那么OSSemPend()函数对该值减1并返回，如果调用时信号量等于0，那么OSSemPend()函数将任务加入该信号量的等待列表，任务将等待直到获得信号量或者超时。OSSemPend()函数代码大体如下：

int OSSemPend(int i, int priority)

{

char OSMapTbl[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80}; //定义优先级映射表

OSEvent \*p3= Semlist->next;

while(p3->number != i)

{

P3=p3->next;

}

if(!p3)

{

return 0;

} //如果不在ISR中，则错误，返回0。

, if(p3->OSEventCnt > 0)

{

P3->OSEventCnt --;

return 1; //如果信号量的值大于0，,对该值减1并返回

}

else

{

p3->OSEventGrp |= OSMapTbl[priority>>3];

p3->OSEventTbl[priority >>3] |= OSMapTbl[priority & 0x07];//如果信号量的值小于0或者等于0，将当前任务从就绪队列里出来

OSschedule(6);//任务调度

}

}

（3）任务获得信号量并在访问共享资源后必须释放信号量，释放信号量及发送信号量。获得信号量函数判断事件为任务量后，再判断是否有任务等待信号量，当确定有任务等待信号量时，使相应的任务就绪，再进行任务调度。没有任务等待信号量时，信号量计数器加1。OSSemPost()函数的代码大体如下：

int OSSemPost(int i,int priority)

{

int self = 6; //当前正在运行任务的优先级

Int high,low;

OSEvent \*p = Semlist->next;

If(p=NULL)

return 0; //ECB指针为NULL时，错误返回0，p2是生成信号量时建立的指针

if(p->OSEventGrp != 0) //判断当前ECB里没有要等的任务

{

//这里获取最高优先级的任务

high = OSMapTbl[OSRdyGrap];

low = OSMapTbl[OSRdyTbl [high]];

priority = (high<<3)+low;

OSschedule(6)//调度

return 1;

}

else

{

p->OSEventCnt ++;

return 1;

}

}

### 四、 总结

通过本次试验，我对操作系统的资源管理过程有了更加深入的认识。认识了事件控制块ECB这一基本数据结构。在本次试验中，我也发现自己有很多不足，代码的逻辑性欠缺,OSMapTbl[ ]等基本结构忘了定义等，这些基本的习惯没有建立是非常不好的。在以后的学习和实验中，要时刻注意细节，不断学习和进步。