**UCOSII在STM32上的移植**

1. CM3的寄存器组
2. 通用寄存器R0—R12。函数返回时返回值存在R0中。
3. 堆栈指针寄存器R13。堆栈指针寄存器分为MSP和PSP。MSP可以在异常和应用程序中使用，PSP只能在应用程序中使用。复位后，处理器默认进入线程模式，特权极访问，并使用MSP堆栈指针寄存器。
4. 连接寄存器R14（又名LR寄存器）。作用是存储返回地址。
5. 程序计数器R15（又名PC寄存器）。
6. 特殊功能寄存器
7. 程序状态字寄存器组PSR。程序状态字寄存器组分为：
8. 应用程序状态寄存器APSR：0 标志，进位标志，负数标志，溢出标志

2）中断号IPSR：记录当前的中断号

3）执行状态寄存器 EPSR：

这三寄存器都是 32 位，但由于它们的有效位是错开的，因此可以通过“PSR”或者“xPSR”这两个名字来同时访问三个寄存器。

1. 中断屏蔽寄存器组PRIMASK, FAULTMASK, BASEPRI
2. PRIMASK：除了NMI和硬件fault以外，除能所有的中断。
3. FAULTMASK：除了NMI以外，除能所有的中断。
4. BASEPRI：除能所有优先级不高于某个具体数值的中断。
5. 控制寄存器CONTROL。控制寄存器有两个作用：

1）定义特权级别

2）选择当前使用哪个堆栈指针

ucosii 不区分特权级和用户级，程序始终工作在特权级。

1. 函数调用及中断产生时寄存器的入栈情况

当产生中断时，CM3硬件内核会自动依次入栈xPSR,PC,LR,R12,R3,R2,R1,R0这8个寄存器。而R4—R11这几个寄存器如果在中断服务函数中被使用到了，则编译器会在使用该寄存器前添加入栈代码，使用完后添加出栈代码。

在ucosii中，PendSV\_Handler异常服务函数是用汇编写的，所以编译器无法加入R4—R11的入栈出栈代码（即使添加了入栈出栈的代码，也不是ucosii任务切换想要的效果），必须手动编写入栈出栈的代码。且入栈的是被中断的任务的R4—R11寄存器，入栈的位置是被中断任务的任务堆栈。出栈的是当前就绪的最高优先级任务的R4—R11寄存器，出栈的位置是当前就绪的最高优先级任务的任务堆栈。

1. 从异常中返回

在进入异常服务程序后，LR的值被自动更新为特殊的EXC\_RETURN，这是一个高28位全为1的值，只有[3:0]的值有特殊含义，如下表所示。

|  |  |
| --- | --- |
| 段位 | 含义 |
| 【31:4】 | EXC\_RETURN的标识：必须全为1 |
| 3 | 0：返回后进入Handler模式  1：返回后进入线程模式 |
| 2 | 0：返回后使用MSP从主堆栈中做出栈操作。  1：返回后使用PSP从进程堆栈中做出栈操作。 |
| 1 | 保留，必须为0 |
| 0 | 0：返回ARM状态。  1：返回Thumb状态。 在CM3 中必须为1 |

在PendSV\_Handler异常服务函数返回前要将当前就绪的最高优先级任务的堆栈指针赋给PSP，然后置位LR寄存器的位2，这样CPU就会使用PSP从任务堆栈中出栈，从而切换到对应的任务上。

1. ucosii任务创建，以及移植时要编写os\_cpu\_c.c文件中的OSTaskStkInit()函数：

在OSTaskCreat()函数中主要做以下几件事：

1. 检查是否在中断里，如果在中断里则不允许创建任务。
2. 检查任务优先级是否被占用了，如果优先级已经被占用则无法创建任务。
3. 将OSTCBPrioTbl[]中对应的优先级置1，表明这个优先级已经被占用。
4. 调用OSTaskStkInit()函数来初始化任务堆栈。
5. 调用OS\_TCBInit()函数来初始化任务控制块。
6. 触发一次任务调度。

Ucosii的任务调度方式是通过软件触发PendSV异常，在异常服务函数中手动将R4—R11寄存器入栈到被中断的任务的堆栈上。然后从当前就绪的最高优先级任务的堆栈里手动出栈R4—R11寄存器。接着把当前最高优先级任务的堆栈地址赋给PSP堆栈指针，然后置位LR寄存器的位2，使异常函数返回后使用PSP堆栈指针。异常函数退出时，硬件自动从当前的SP寄存器指向的地址处出栈其他寄存器。这样就完成了一次任务调度。

创建任务时必须手动将任务的重要数据入栈（详细请参考OSTaskStkInit()函数）：

1. xPSR寄存器的值必须为0x01000000
2. PC的值为任务的函数指针
3. LR寄存器的值必须为0xFFFFFFFEL
4. R0 存放任务函数的参数
5. 其它寄存器写入任意值
6. os\_cpu\_a.s文件中的OSStartHighRdy汇编函数。

OSStartHighRdy函数在OSStart()函数中被调用，这个函数的作用是启动ucosii的第一个任务。

.thumb\_func

OSStartHighRdy:

@设置PendSV异常中断的优先级为最小优先级

LDR R0, =NVIC\_SYSPRI14

LDR R1, =NVIC\_PENDSV\_PRI

STRB R1, [R0]

@设置PSP指针为0，这样在PendSV\_Handler函数中就不会重复入栈R4—R11寄存器

MOVS R0, #0

MSR PSP, R0

@将全局变量OSRunning的值设为1，表明ucosii已经运行。

LDR R0, =OSRunning

MOV R1, #1

STRB R1, [R0]

@触发PendSV中断

LDR R0, =NVIC\_INT\_CTRL

LDR R1, =NVIC\_PENDSVSET

STR R1, [R0]

@开中断

CPSIE I

1. os\_cpu\_a.s文件中的PendSV\_Handler汇编函数：

.thumb\_func

PendSV\_Handler:

CPSID I

@如果PSP的值为0则说明是运行ucosii的第一个任务，则跳过保存R4--R11寄存器

MRS R0, PSP

CBZ R0, OS\_CPU\_PendSVHandler\_nosave

@保存被中断的任务的R4--R11寄存器到该任务的任务堆栈中。

SUBS R0, R0, #0x20

STM R0, {R4-R11}

@更新被中断任务的任务控制块的OSTCBStkPtr成员为当前的PSP的值。

@OSTCBCur是一个全局变量指针，总是指向当前运行任务的任务堆栈。

LDR R1, =OSTCBCur

LDR R1, [R1]

STR R0, [R1]

OS\_CPU\_PendSVHandler\_nosave:

@调用OSTaskSwHook()函数。

PUSH {R14}

LDR R0, =OSTaskSwHook

BLX R0

POP {R14}

@设置全局变量OSPrioCur=OSPrioHighRdy。

@OSPrioCur是一个全局变量，表示当前运行的任务的优先级。

LDR R0, =OSPrioCur

LDR R1, =OSPrioHighRdy

LDRB R2, [R1]

STRB R2, [R0]

@设置全局变量OSTCBCur = OSTCBHighRdy。

LDR R0, =OSTCBCur

LDR R1, =OSTCBHighRdy

LDR R2, [R1]

STR R2, [R0]

@从当前就绪的最高优先级任务的任务堆栈中出栈R4--R11。

LDR R0, [R2]

LDM R0, {R4-R11}

@R0里的值为：当前就绪的最高优先级任务的任务堆栈中保存R0值的位置。

ADDS R0, R0, #0x20

@把R0里的值赋给PSP。当退出PendSV\_Handler函数后，CM3内核会自动从PSP指向堆栈地址处取出xPSR,PC,LR,R12,R3,R2,R1,R0这8个寄存器的值，并赋给相应寄存器。

MSR PSP, R0

@置位LR寄存器的位2，使异常函数返回后使用PSP堆栈指针。

ORR LR, LR, #0x04

CPSIE I

BX LR

1. os\_cpu\_a.s文件中的OS\_CPU\_SR\_Save函数：

.thumb\_func

OS\_CPU\_SR\_Save:

MRS R0, PRIMASK @读取PRIMASK到R0,R0为函数的返回值

CPSID I @PRIMASK=1,关中断(NMI和硬件FAULT可以响应)

BX LR @返回

1. os\_cpu\_a.s文件中的OS\_CPU\_SR\_Restore函数：

.thumb\_func

OS\_CPU\_SR\_Restore:

MSR PRIMASK, R0 @读取R0到PRIMASK中,R0为函数的第一个参数

BX LR @返回

1. os\_cpu\_a.s文件中的OSCtxSw函数：

.thumb\_func

OSCtxSw:

PUSH {R4, R5}

LDR R4, =NVIC\_INT\_CTRL @触发PendSV异常 (causes context switch)

LDR R5, =NVIC\_PENDSVSET

STR R5, [R4]

POP {R4, R5}

BX LR

1. os\_cpu\_a.s文件中的OSIntCtxSw函数：

.thumb\_func

OSIntCtxSw:

PUSH {R4, R5}

LDR R4, =NVIC\_INT\_CTRL @触发PendSV异常 (causes context switch)

LDR R5, =NVIC\_PENDSVSET

STR R5, [R4]

POP {R4, R5}

BX LR

NOP

**UCOSII实时操作系统**

1. 实时操作系统RTOS（Real time operate system）是指当外部事件或数据产生时，能够以足够快的速度予以处理，其处理的结果又能在规定的时间内来控制生产过程或对处理系统做出快速响应，并使所有任务协调一致的操作系统。
2. 实时操作系统的特点：
3. 实时操作系统是多任务的。
4. 实时操作系统的内核是可剥夺的。
5. 进程调度的延时必须可预测并且尽可能小。
6. 系统的服务时间是可知的。
7. 中断延时尽可能小。
8. 代码临界区（critical section）

虽然多任务系统中很多资源是共享的，但某些共享资源一次只允许一个任务访问，这样的资源称为临界资源。临界区就是指每个任务访问共享资源的那部分代码。

1. 优先级反转

优先级反转是实时操作系统最常见的问题。如，存在3个任务，任务1优先级比任务2优先级高，任务2优先级比任务3优先级高。任务1和任务2处于等待状态，任务3正在运行。如果此时任务3正在使用共享资源，而任务1等待时间结束并剥夺了任务3的CPU控制权，但是任务1正好要使用被任务3占用的共享资源，此时任务1无法得到共享资源而放弃CPU控制权，让任务3继续运行。此时任务2等待时间结束并剥夺了任务3的CPU控制权并运行。原本任务2的优先级要小于任务1，但是任务1由于被任务3阻塞而使任务2比任务1优先得到运行，这种情况就是优先级反转。

防止优先级反转的方法是：当任务1获得共享资源失败时，先将任务3的优先级提高到和任务1一样，然后任务1再放弃CPU控制权。这样优先级低于任务1低的任务就不会优先得到运行。当任务3使用完共享资源后将自己的优先级恢复成原来的优先级。在UCOS中使用OSTaskChangePrio()函数来改变任务的优先级。

1. 时间片轮转调度

当两个或者两个以上的任务拥有相同的优先级时，内核允许一个任务运行一个固定时间以后将CPU控制权切换给另一个相同优先级的任务，这被称为时间片调度。

在UCOSII中每个任务的优先级都是唯一的，不同任务间不能使用同一个优先级，所以UCOSII没有时间片轮转调度的功能。而UCOSIII不同任务可以使用同一个优先级，相同优先级的任务间可以进行时间片轮转调度。

1. UCOS中任务的同步方式有：信号量、消息队列、事件标志组和邮箱。
2. UCOSII的全局变量

|  |  |
| --- | --- |
| 变量 | 变量说明 |
| OSPrioCur | 类型为INT8U，正在运行的任务优先级 |
| OSPrioHighRdy | 类型为INT8U，具有最高优先级的就绪任务的优先级 |
| OSTCBCur | 类型为OS\_TCB \*，指向正在运行的任务的任务控制块 |
| OSTCBHighRdy | 类型为OS\_TCB \*，指向最高优先级就绪任务的任务控制块 |
| OSTime | 类型为INT32U，表示系统当前时间（节拍数） |
| OSIntNesting | 类型为INT8U，存放中断嵌套的层数（0~255） |
| OSLockNesting | 类型为INT8U，调用了OSSchededLock的嵌套层数 |
| OSCtxSwCtr | 类型为INT32U，上下文切换的次数 |
| OSTaskCtr | 类型为INT8U，已建立的任务数 |
| OSRunning | 类型为BOOLEAN，表明内核是否已经运行 |
| OSCPUUsage | 类型为INT8S，存放CPU的利用率（%） |
| OSIdleCtrMax | 类型为INT32U，表示每秒空闲任务计数的最大值 |
| OSIdleCtrRun | 类型为INT32U，表示空闲任务计数器每秒的计数值 |
| OSIdleCtr | 类型为INT32U，空闲任务的计数器 |
| OSStatRdy | 类型为BOOLEAN，统计任务是否就绪的标志 |
| OSIntExit | 类型为INT8U，用于函数OSInitExt() |

1. UCOS的5种任务状态：

|  |
| --- |
| UCOS的5种任务状态 |
| 睡眠：该任务驻留在内存中，但不被多任务内核所调用。当没有使用OSTaskCreate()函数来创建任务，以及任务被剥夺了任务控制块的时的状态就是随眠状态。 |
| 就绪：新任务已经分配到了任务控制块，并且具备了运行的条件，但由于新任务的优先级比当前任务低，此时新任务就处于就绪状态。 |
| 等待：任务调用延时函数或者等待信号量、消息、事件标志组时会进入等待状态，让出CPU控制权。 |
| 运行：任务获得CPU控制权并运行任务服务例程。 |
| 被中断：一个正在运行的任务被异常和中断剥夺CPU控制权。 |

1. 任务控制块OS\_TCB：

主要用于保存任务的各种信息，如任务堆栈指针、栈的大小、任务状态、任务优先级等。其中对于用户程序来说，比较有用的是OSTCBStat成员，OSTCBStat的可能的值为：

|  |  |
| --- | --- |
| OS\_STAT\_RDY | 表示任务处于就绪状态 |
| OS\_STAT\_SEM | 表示任务处于等待信号量状态 |
| OS\_STAT\_MBOX | 表示任务处于等待邮箱状态 |
| OS\_STAT\_Q | 表示任务处于等待消息队列状态 |
| OS\_STAT\_SUSPEND | 表示任务处于被挂起状态 |
| OS\_STAT\_MUTEX | 表示任务处于等待互斥量状态 |

1. 就绪表

UCOSII：分为OSRdyGrp和OSRdyTbl[]

UCOSIII：分为OSPrioTbl[]和OSRdyList[]

1. UCOS的调度：

任务调度：OSSched()

中断调度：OSIntExit()

1. UCOSIII的系统内部任务：

|  |
| --- |
| 空闲任务：所有任务中优先级最低的任务。 |
| 时钟节拍任务：时钟节拍任务的优先级要尽可以设得高一些，一般设为1。 |
| 定时任务：定时任务的优先级尽可能设得高一些，但必须比时钟节拍任务要低，一般设为2。 |
| 中断服务管理任务（UCOSII有，UCOSIII无）：优先级必须为最高优先级0。硬件中断可以剥夺任何任务的CPU使用权，所以中断服务管理任务的作用不是为了调用中断服务函数，而是为了延迟在ISR中的post类函数对信号量的释放或消息的发送等。如果使能了中断服务管理任务，则在ISR中调用post类函数时，要发送的数据和数据地址会被缓存起来，当所有嵌套ISR都退出后，中断服务管理任务会优先执行，将数据发送出去。 |
| 统计任务：作用是统计各个任务的CPU使用率和堆栈使用情况，它的优先级必须设置为倒数第2低，只比空闲任务的优先级要高。 |
| 钩子函数 |

1. 判断UCOS是否已经运行：

if(OSRunning == OS\_STATE\_OS\_RUNNING)

{

Printf(“ucos已经运行”);

}

1. 使用UCOS后中断函数入口处必须使用OSIntEnter()函数，出口处必须使用OSIntExit()函数

OSIntEnter：判断OS是否已经启动；判断中断嵌套数是否超过限制，如果未超过限制则加1。

OSIntExit：中断嵌套数减1，然后触发PendSV异常。

1. 临界区代码保护

宏OS\_CFG\_ISR\_POST\_DEFERRED\_END定义为0时，使用开关中断的方式保护临界区，当定义为1时使用给调度器上锁的方式来保护临界区代码。

OS\_CRITICAL\_ENTER()：进入临界区

OS\_CRITICAL\_EXIT()：退出临界区

OS\_CRITICAL\_EXIT\_NO\_SCHED()：退出临界区

1. 时间管理函数

OSTimeDly()、OSTimeDlyHMSM()： 延时函数

OSTimeDlyResum()：用于恢复由于调用了延时函数而片于等待状态的函数，使之恢复运行。

|  |  |  |
| --- | --- | --- |
| 延时函数的的4种延时方式 | | |
| OS\_OPT\_TIME\_DLY | 相对模式 | 延时误差会积累 |
| OS\_OPT\_TIME\_TIMEOUT | 相对模式 | 延时误差会积累 |
| OS\_OPT\_TIME\_MATCH | 绝对模式 | 从开机到指定的时间后调用任务函数 |
| OS\_OPT\_TIME\_PERIODIC | 周期模式 | 延时误差不会积累 |

1. 任务的挂起和恢复

OSTaskSupend：挂起任务

OSTaskResume：恢复被挂起的任务

1. 信号量：信号量的作用是用于等待某个事件的发生，用于任务同步。

|  |  |
| --- | --- |
| 信号量主要函数 | |
| OSSemCreate() | 创建一个信号量，可以设置获得信号量的任务数量。 |
| OSSemDel() | 删除一个信号量 |
| OSSemPend() | 等待一个信号量，可以设置等待超时 |
| OSSemPendAbort() | 一个任务取消其它任务对信号量的等待 |
| OSSemPost() | 释放信号量，让其它任务获得信号量 |
| OSSemSet | 设置信号量的值 |

1. 互斥量：

UCOS中的互斥量不仅仅是对任务上锁进行互斥，更重要的是为了防止优先级反转。

如果某个低优先级的任务拥有互斥量，高优先级的任务抢占CPU以后发现互斥量被低优先级的任务占有了，则系统会将占有互斥量的任务的优先级提升至和当前任务一样，然后放弃CPU控制权等待信号量。拥有互斥量的任务断续运行，直到不再需要互斥量而释放互斥量。释放互斥量时系统会将被提升的任务的优先级恢复成原本的优先级。这样就有效防止了由于低优先级任务对共享资源的占用而阻塞高优先级任务，使得中等优先级任务和高优先级任务之间的优先级反转。

|  |  |
| --- | --- |
| 互斥量主要函数 | |
| OSMutexCreate() | 创建互斥量 |
| OSMutexDel() | 删除互斥量 |
| OSMutexPend() | 等待互斥量 |
| OSMutexPendAbord() | 取消对互斥量的等待 |
| OSMutexPost() | 释放互斥量让其它任务获得互斥量 |

1. 消息队列

消息队列主要包含三个部分：

1. 指向数据的指针
2. 表明数据长度的变量
3. 记录消息发送时间的时间戳

由于消息的传送是通过指针来传递的，而不会复制数据内容，所以消息指针指向的数据必须一直保持可见性，即传送的数据必须是全局变量、局部静态变量、函数指针或者通过malloc申请的内存。

|  |  |
| --- | --- |
| 消息队列主要函数 | |
| OSQCreate() | 创建消息队列 |
| OSQDel() | 删除消息队列 |
| OSQFlush() | 清空消息队列里的消息 |
| OSQPend() | 等待消息 |
| OSQPendAbort() | 取消对消息的等待 |
| OSQPost() | 向消息队列发送一个消息 |

1. 任务内嵌信号量和消息队列

在UCOSIII中每个任务都内嵌了信号量和消息队列，使用任务内嵌的信号量和消息队列能简化代码，并使程序更有效。

|  |  |
| --- | --- |
| 任务内嵌信号量主要函数 | |
| OSTaskSemPend() | 等待任务信号量 |
| OSTaskSemPendAbort() | 取消对任务信号量的等待 |
| OSTaskSemPost() | 向任务发布信号量 |
| OSTaskSemSet() | 设置信号量计数值 |

|  |  |
| --- | --- |
| 任务内嵌消息队列主要函数 | |
| OSTaskQPend() | 等待任务消息 |
| OSTaskQPendAbort() | 取消等待任务消息 |
| OSTaskQPost() | 向任务发送一条消息 |
| OSTaskQFlush() | 清空任务消息队列里的消息 |

1. 事件标志组

事件标志组用来解决一个任务和多个任务同步的问题，事件标志组同步方式分为与同步和或同步。当任意一个事件发生时，任务都被同步的同步方式是或同步；当所有事件都发生后任务才被同步的同步方式是与同步。

1. 创建事件标志组的函数：

void OSFlagCreate(OS\_FLAG\_GRP \*p\_grp,

CPU\_CHAR \*p\_name,

OS\_FLAGS flags,

OS\_ERR \*p\_err)

参数p\_grp：指向事件标志组的指针

参数p\_flalgs：设置事件标志组的初始值

1. 等待事件标志组的函数：

OS\_FLAGS OSFlagPend(OS\_FLAG\_GRP \*p\_grp,

OS\_FLAGS flags,

OS\_TICK timeout,

OS\_OPT opt,

CPU\_TS \*p\_ts,

OS\_ERR \*p\_err)

参数flags：等待序列。任务需要等待事件标志组中哪些事件，就把flags中对应位置位，不管等待的是置位事件还是复位事件，都应该把flags中对应位置位。

参数opt：决定同步方式是所有等待位置1、所有等待位置0、任意一等待位置1或者任意一等待位置0。

参数opt的可选项为如下：

OS\_OPT\_FLAG\_CLR\_ALL：所有等待位复位

OS\_OPT\_FLAG\_CLR\_ANY：任意等待位复位

OS\_OPT\_FLAG\_SET\_ALL：所有等待位置位

OS\_OPT\_FLAG\_SET\_ANY：任意等待位复位

1. OS\_FLAGS OSFlagsPost(OS\_FLAG\_GRP \*p\_grp,

OS\_FLAGS flags,

OS\_OPT opt,

OS\_ERR \*p\_err)

参数flags：选择要对事件标志组中哪些位进行操作。

参数opt：对选择的位进行置位操作或复位操作。

opt可选的项为：

OS\_OPT\_POST\_FLAG\_SET：置位操作

OS\_OPT\_POST\_FLAG\_CLR：复位操作

1. 同时等待多个内核事件

UCOSIII中支持一个任务同时等待多个信号量、多个消息队列，但是不支持同时等待多个互斥量和多个事件标志组。

等待多个内核事件的函数：

OS\_OBJ\_QTY OSPendMuti(OS\_PEND\_DATA \*p\_pend\_data\_tbl,

OS\_OBJ\_QTY tbl\_size,

OS\_TICK timeout,

OS\_OPT opt,

OS\_ERR \*p\_err)

等待多个内核事件的一个例子：

OS\_PEND\_DATA pend\_tbl[3];

pend\_tbl[0].PendObjPtr = (OS\_PEND\_OBJ \*)&sem1; //等待信号量1

pend\_tbl[0].PendObjPtr = (OS\_PEND\_OBJ \*)&sem2; //等待信号量2

pend\_tbl[0].PendObjPtr = (OS\_PEND\_OBJ \*)&Queue; //等待消息队列

while(1)

{

index = OSPendMuti((OS\_PEND\_DATA \*) pend\_tbl,

(OS\_OBJ\_QTY) sizeof(pend\_tbl)/sizeof(pend\_tbl[0]),

(OS\_TICK)0,(OS\_OPT) OS\_OPT\_PEND\_BLOCKING,

(OS\_ERR \*) p\_err);

do\_something(index);

}