第33卷 第13期 计 算 机 工 程 2007年7月
Vol.33 No.13 Computer Engineering July 2007
·软件技术与数据库· 文章编号：1000—3428(2007)13—0099—02 文献标识码：A 中图分类号：TP311

uC/OS-Ⅱ任务数扩充的理论与实现

阚宏伟[1]，马小平[1]，杜 林[2]
(1. 中国矿业大学计算机学院，徐州 221008；2. 安徽电力继远软件公司技术支持中心，合肥 230000)

摘 要：结合当前流行的uC/OS-Ⅱ v2.52对任务的管理形式，推导出了几种可用任务数扩充方案。给出了一个在ARM核MCU上运行效率较优的最小系统实例。在ARM7 MCU LPC2294上对改进后的操作系统进行资源消耗分析实验，得出时钟周期消耗数据，验证了该方案是一种比较理想的解决方案。

关键词：RTOS；uC/OS-Ⅱ；任务扩充；LPC2294；ARM

Theory and Implementation of Increasing the Number of Tasks in uC/OS-Ⅱ

KAN Hongwei[1], MA Xiaoping[1], DU Lin[2]
(1. College of Computer, China University of Mining and Technology, Xuzhou 221008;
2. R&D Center, Anhui Dianli Jiyuan Software Co. Ltd., Hefei 230000)

【Abstract】Based on the uC/OS-Ⅱ v2.52 which is task management system, this paper presents solutions for increasing the number of usable tasks. An example of the least RTOS is given under the ARM MCU. Through the clock cycles from an experiment under the ARM7 MCU LPC2294, the solution is confirmed feasible.

【Key words】RTOS; uC/OS-Ⅱ; expansion task; LPC2294; ARM

uC/OS-Ⅱ v2.52作为一个通过“美国联邦航空管理局(FAA)对用于商业飞机的、符合RTCA DO178B标准认证[1]”的RTOS，其安全性和稳定性应该比其他大多数实时嵌入式优一些。随着嵌入式系统应用的深入，uC/OS-Ⅱ对任务数最高64个的限制，有时会在相对复杂和面向对象的系统开发过程中，成为一个瓶颈资源。本文就这个问题，结合自己的项目经验，提出对uC/OS-Ⅱ可用任务数的增加方案，加以实现和分析。

在uC/OS-Ⅱ中，系统调度器对任务的调度通过任务就绪表OSRdyTbl[](8*8=64bit)来实现。为提高查找最高优先级任务的速度，调度器引入了几个变量OSRdyGrp、OSMapTBL[]、OSUnMapTbl[]。

1 对任务数扩充的理论方案

在嵌入式实时操作系统中，经常包含一些决定整个系统性能的关键代码，如果处理不好，就会显著的增加系统对资源消耗。而响应时间对RTOS是一个致命的性能指标，所以对系统内核的修改算法应该尽可能的优化，不然没有实际价值。uC/OS-Ⅱ在编写的时候必须兼顾8位的单片机。虽然64个任务对8位机来说应该是够了，但目前能用到更多任务数的MCU几乎都是16位以上的，这也当然包括ARM核的MCU。

鉴于uC/OS-Ⅱ v2.52对任务的管理方式和ARM体系结构已经比较成熟，扩充方案至少必须考虑到下面几点：

(1)应尽可能保持uC/OS原来的系统结构优势，充分考虑软件的复用和移植的方便，所以代码的修改量应该越小越好。

(2)系统中任务的调度非常频繁，而且是占用的系统绝对优势资源，像片内高速缓存和ALU的计算能力，所以对应的代码不能占用太多存储空间，涉及的变量也要尽量少。

(3)充分考虑ARM核MCU的下面3个特点：1)ARM处理器的一个显著特点就是，可以在操作数进入ALU之前，对其进行左右移位，这显著增强了对许多数据处理的操作灵活性[3]。预处理或移位发生在该指令周期内[4]。2)8位、16位的数据在装载/存储ARM寄存器之前，先要扩展成32位。这就意味着装载更小数据并不节省指令时间。3)ARM核具有高效的带有条件执行功能的指令。

保持原来的结构意味着，如果增加可用的任务数，表示任务状态的OSRdyTbl[]数组的位数或下标必须增加；OSRdyGrp位数、OSMapTBL[]的下标应该随着OSRdyTbl[]的下标的增长而增长。这种增加是线性关系的，也是必须的。比如就绪表，按照系统的结构框架每位只有两种状态，8*8的结构必须增加。但所有增加的幅度不能超过32位。涉及到OSUnMapTbl[]的下标，如果要满足16位的增长应该是2^{16}=65 536个，如果32位就是2^{32}，这种指数级的空间增长显然不可行。所以只能用运算能力来换取对这个资源的需求。用最少的运算能力，来解决这个问题的算法分析是本文的要点。

对OSRdyTbl[]的位数和下标的增长组合如下：32*32＝1024、16*32＝512、16*16＝256、8*16＝128。实际的项目需求任务一般很难超过256个，应该以最大128个居多。但扩充的原理是一样的，这里以128为例加以分析，256的方案原理是一样的。实现时用128的方案为例，256的方案实现

作者简介：阚宏伟(1975－)，男，硕士，主研方向：嵌入式智能控制；马小平，教授、博士生导师；杜 林，学士

收稿日期：2006-08-10 **E-mail：**Orien_kan@163.com

时略有不同。

2 对任务数扩充的实现

本次扩充采用的是 uC/OS-Ⅱ v2.52 的一个最小系统，完整的扩充方案笔者会在另外一篇论文中给出。最小系统配置如表 1 所示(斜体表示改动部位，且只列出需改动语句，下同)。

表 1 在文件 OS_CFG.H 中的改动

#define OS_LOWEST_PRIO	*127*
#define OS_TASK_STAT_EN	*0*
#define OS_FLAG_EN	*0*
#define OS_MBOX_EN	*0*
#define OS_MEM_EN	*0*
#define OS_MUTEX_EN	*0*
#define OS_Q_EN	*0*
#define OS_SEM_EN	*0*
#define OS_TASK_CHANGE_PRIO_EN	*0*
#define OS_TASK_DEL_EN	*0*
#define OS_TASK_QUERY_EN	*0*
#define OS_SCHED_LOCK_EN	*0*

根据 ARM 核的第 2 个特点，对 OSRdyTbl[]的扩容应该首先利用位数。这里选择将位数扩到 16 位，下标数不变 8 个。OSMapTbl[]下标和位数扩到 16 个，其他变量基本不需变动，整体调整很小。如表 2 所示。

表 2 在文件 Ucos_II.H 中的改动

```
#define OS_EVENT_TBL_SIZE   ((OS_LOWEST_PRIO) / 16 + 1)
#define OS_RDY_TBL_SIZE     ((OS_LOWEST_PRIO) / 16 + 1)

INT16U              OSTCBBitX;      // in TASK CONTROL BLOCK
INT16U              OSTCBBitY;

OS_EXT  INT16U  OSRdyTbl[OS_RDY_TBL_SIZE];   // in GLOBAL
extern  INT16U  const       OSMapTbl[];      // VARIABLES

#if OS_MAX_TASKS > 127
#error  "OS_CFG.H,      OS_MAX_TASKS must be <= 127"
#endif //in TASK MANAGEMENT of the " LOOK FOR MISSING
#define CONSTANTS "
```

这样系统对存储空间消耗增长为 34B，16 位数据宽度时仅为 8 个内存单元(ARM 系统时多数情况)。这完全在系统的承受能力之内。

因为任务在初值的时候，OSTCBCur->OSTCBX 和 OSTCBCur->OSTCBY 这两个掩码都是一次性计算出来的，将任务置为就绪态和脱离就绪态仅需调整⑤(见表 3)即可。从就绪表中得到优先级数最高的就绪任务的算法②、③、④，也应充分考虑 ARM 核的第 1 特点和第 3 特点，尽量多地选用 if 判断和移位运算。如表 3 所示。

表 3 在文件 OS_CORE.C 中算法的调整

```
INT16U  const  OSMapTbl[] = {0x0001, 0x0002, 0x0004, 0x0008,     // ①
                             0x0010, 0x0020, 0x0040, 0x0080,
                             0x0100, 0x0200, 0x0400, 0x0800,
                             0x1000, 0x2000, 0x4000, 0x8000};
if((OSRdyTbl[OSIntExitY]&0xFF)==0)   // in OSIntExit (void)          ②
        x       =       OSUnMapTbl[OSRdyTbl[OSIntExitY]>>8] + 8 ;
else
        x       =       OSUnMapTbl[OSRdyTbl[OSIntExitY] &0xFF] ;
OSPrioHighRdy = (INT8U)((OSIntExitY << 4) + x));

if      ((OSRdyTbl[y]&0xFF)==0)   // in OSStart (void)              ③
OSPrioHighRdy=(INT8U)((y<<4)+(OSUnMapTbl[OSRdyTbl[y]>>8] + 8)) ;
else
OSPrioHighRdy=(INT8U) ((y<<4)+OSUnMapTbl[OSRdyTbl[y] &0xFF]) ;

if      ((OSRdyTbl[y]&0xFF)==0)   // in OS_Sched (void)             ④
OSPrioHighRdy=(INT8U)((y<<4)+(OSUnMapTbl[OSRdyTbl[y]>>8] + 8)) ;
else
OSPrioHighRdy=(INT8U) ((y<<4)+OSUnMapTbl[OSRdyTbl[y] &0xFF]) ;

ptcb->OSTCBY    = prio >> 4; // in OS_TCBInit()                     ⑤
    Ptcb->OSTCBX    = prio & 0x0F;
```

进行完上面的调整，系统就可以运行优先级为 0~127 间的任务了。

3 扩充方案的资源消耗分析

实验中选用了 PHILIPS 的 LPC2294 MCU 芯片，它的内核是典型的 ARM7TDMI-S ™，具有很好的代表性。因为大系统经常在片外存储器里运行，用片外内 RAM 比较接近现场，加上本文的重点是相对的消耗变化，本次实验用外部 RAM(已验证与片内 RAM 运行数据的相对变化一致，不再熬述)。为了比较清楚地了解扩充方案对 uC/OS-Ⅱ任务调度的影响，用 ARM 公司的 ARM Developer Suite v1.2 作为编译环境，来观察程序改变前后汇编指令的变化。并且以 MCU 系统时钟为计数单位，来精确计算对 ALU 计算能力的消耗。显然需要重点看“从就绪表中得到优先级数最高的就绪任务”的算法，它的变化比较大且被频繁调度。如 OS_CORE.C 中的 OSIntExit ()和 OS_Sched ()函数，应该属于被调度最为频繁的代码之列。

3.1 ARM 汇编结果实验数据

(1)得到最高优先级的就绪任务算法。程序改变前、后的 ARM 汇编指令如表 4 所示。

表 4 程序改变前、后的 ARM 汇编指令 1

ARM 汇编 (64 tasks)	ARM 汇编 (128 tasks)	

ldr r12,0x00000b5c
ldrb r12,[r12,r1]
Ldrb r12,[r14,r12]
	ldr r1,0x00000c00 ; = #0x80000838
	add r1,r1,r0,lsl #1
	ldrh r1,[r1,#0]
	tst r1,#0xff
	bne 0x8f0	

Add r1,r12,r1,lsl #3 and r1,r1,#0xff Strb r1,[r0,#6]	ldrb r1,[r3,r1,lsr #8] add r0,r1,r0,lsl #4 add r0,r0,#8 strb r0,[r2,#6] b 0x900	and r1,r1,#0xff ldrb r1,[r3,r1] add r0,r1,r0,lsl #4 strb r0,[r2,#6]

(2)将任务置为就绪态的算法。程序改变前、后的 ARM 汇编指令如表 5 所示。

表 5 程序改变前、后的 ARM 汇编指令 2

ARM 汇编 (64 tasks)	ARM 汇编 (128 tasks)

ldr r14,0x00000fc8 ;
= #0x8000000b
ldrb r3,[r0,#0x23]
ldrb r12,[r14,#0]
orr r3,r3,r12
strb r3,[r14,#0]
	ldr r14,0x00001074 ;
	= #0x8000000b
	ldrh r3,[r0,#0x24]
	ldrb r12,[r14,#0]
	orr r3,r3,r12
	strb r3,[r14,#0]

ldrb r3,[r0,#0x21]
ldr r12,0x00000fcc ;
= #0x80000038
ldrb r0,[r0,#0x22]
ldrb r12,[r3,r12]!
orr r0,r12,r0
strb r0,[r3,#0]
	ldrb r3,[r0,#0x21]
	ldr r12,0x00001078 ;
	= #0x80000838
	ldrh r0,[r0,#0x22]
	add r3,r12,r3,lsl #1
	ldrh r12,[r3,#0]
	orr r0,r12,r0
	strh r0,[r3,#0]

(3)将任务脱离就绪态的算法。程序改变前、后的 ARM 汇编指令如表 6 所示。

表 6 程序改变前、后的 ARM 汇编指令 3

ARM 汇编 (64 tasks)	ARM 汇编 (128 tasks)

ldrb r1,[r0,#0x21]
ldr r2,0x00000fc8 ;
= #0x80000038
ldrb r3,[r0,#0x22]
ldrb r2,[r1,r2]!
bics r2,r2,r3
strb r2,[r1,#0]
bne 0xf84 ; (OSTaskSuspend + 0xcc)
	ldrb r1,[r0,#0x21]
	ldr r2,0x00001074 ;
	= #0x80000838
	ldrh r3,[r0,#0x22]
	add r1,r2,r1,lsl #1
	ldrh r2,[r1,#0]
	bics r2,r2,r3
	strh r2,[r1,#0]
	bne 0x1030
	; (OSTaskSuspend + 0xd0)

ldr r3,0x00000fc4 ;
= #0x8000000b
ldrb r1,[r0,#0x23]
ldrb r2,[r3,#0]
bic r1,r2,r1
strb r1,[r3,#0]
	ldr r3,0x00001070 ;
	= #0x8000000b
	ldrh r1,[r0,#0x24]
	ldrb r2,[r3,#0]
	bic r1,r2,r1
	strb r1,[r3,#0]

从上面汇编结果看，效果相当令人满意。说明上面提供的 C 语句具有很高的执行效率，ADS 利用 ARM 指令的条件执行功能，已经将其优化到只须增加 3 或 4 条指令(if 语句的两个分支)。这里需要说明的是：“ldr”和“str”指令，片外运行时对存储器的依赖较大，在一般的系统中均比较消耗时钟周期。因此“ldr”和“str”指令的多少对时钟消耗影响较大。其他指令平均每个时钟大约执行 0.9 条(根据 ARM 公司的数据)。 （下转第 103 页）