

# $\mu$ C/OS-Ⅱ操作系统在ARM处理器上的移植

王秀丽<sup>1,2</sup>, 麦继平<sup>1</sup>

(1. 天津工业大学 计算机学院, 天津 300160;

2. 天津城建学院 电子与信息工程系, 天津 300384)

**摘要:**当前嵌入式实时操作系统(RTOS)的使用使得实时应用程序的设计和扩展变得容易,不需大改动就可增加新的功能,对时间要求苛刻的事件能快捷、有效地处理。通过有效的系统服务,RTOS使得资源得到更好利用。 $\mu$ C/OS-Ⅱ是一个基于优先级的抢占式实时内核的实时操作系统,程序可读性强,移植性好,代码固定,可裁剪,非常灵活,源码公开,所以得到了广泛应用。介绍了 $\mu$ C/OS-Ⅱ内核的特点、工作原理;通过实验将其在ARM处理器上移植,证明了其较好的移植性。

**关键词:**实时操作系统;嵌入式系统;中断;移植

中图分类号:TP316.2

文献标识码:A

文章编号:1673-629X(2007)01-0213-03

## Replantation of $\mu$ C/OS-Ⅱ Operating System in ARM Processor

WANG Xiu-li<sup>1,2</sup>, MAI Ji-ping<sup>1</sup>

(1. Institute of Computer Tech. and Automatization, Tianjin Polytechnic Univ., Tianjin 300160, China;

2. Dept. of Electronic and Comm. Eng., Tianjin Urban Construction Inst., Tianjin 300384, China)

**Abstract:** Currently, it makes design and extension of the real time application easy that the embedded real time operating system (RTOS) is applied. RTOS can add new function when it needn't be changed more, and it can deal with the real time case effectively. Via effective system serve, RTOS makes use of resource.  $\mu$ C/OS-Ⅱ is a real time operating system based on PRI, occupied and real time kernel. Programme has stronger readability and better replanted nature. The code is fixed and it can be cutted out. So it has very good agility. This paper describes characteristic and work elements of  $\mu$ C/OS-Ⅱ kernel. At the same time, with experimentation, RTOS is replanted to the ARM processor. The experimentaion testifies its better replantated nature.

**Key words:** real time operating system; embedded system; interruption; replantation

### 0 引言

实时操作系统(RTOS)是一段系统启动后首先执行的背景程序,用户应用程序是运行在RTOS上的任务。在嵌入式应用中,使用实时操作系统的最主要原因是提高系统的可靠性,其次是提高开发效率、缩短开发周期。实时操作系统 $\mu$ C/OS-Ⅱ是一个基于优先级的抢占式实时内核,程序可读性强,移植性好,代码固定,可裁剪,非常灵活。ARM公司是全球领先的16/32位RISC微处理器供应商,因为RISC微处理器高性能、低功耗、低成本、小体积,广泛适用于实时环境、开放平台及数字信号处理(DSP)运算等<sup>[1,2]</sup>。

### 1 $\mu$ C/OS-Ⅱ主要特点

$\mu$ C/OS-Ⅱ的主要特点如下<sup>[1,3,4]</sup>:

\* 公开源码:是为数不多的公开源码的RTOS,给二

次开发和移植提供了可能;

\* 可移植性强: $\mu$ C/OS-Ⅱ绝大多数源码用ANSI C编写,少量用汇编语言编写,具有较强的移植性;

\* 可固化:微小内核,可以和应用程序一起固化到FLASHROM中;

\* 可裁剪:通过条件编译即可实现裁剪,十分方便;

\* 占先式:是实时性的重要保证;

\* 多任务:多达64个任务管理,可以满足大多数控制任务;

\* 可确定性:全部的函数调用与服务执行的时间是可知的;

\* 系统服务:提供众多的系统服务,如:消息队列、信号量、内存管理等;

\* 中断管理:多达255层的中断管理;

\* 稳定性和可靠性强。

### 2 $\mu$ C/OS-Ⅱ内核工作原理

#### 2.1 工作原理

在各种多任务RTOS中,任务之间的切换是执行最频繁的,因此任何RTOS任务调度与切换的代码越简洁,系

收稿日期:2006-03-20

作者简介:王秀丽(1977-),女,天津宁河人,硕士研究生,实验师,研究方向为模式识别与智能系统;麦继平,教授,研究方向为控制理论与控制工程。



统的实时性和中断处理敏感性就会越高。

实时嵌入式操作系统  $\mu\text{C}/\text{OS}-\text{II}$  实现任务切换的方式非常小巧,它充分利用了中断返回指令,只有非常少的汇编代码就实现了不同任务的切换,用户需要尽自己所能将堆栈结构设置成与中断堆栈结构一样。

$\mu\text{C}/\text{OS}-\text{II}$  首先在主程序中对操作系统进行初始化,完成  $\mu\text{C}/\text{OS}-\text{II}$  所有变量和数据结构的初始化,包括任务控制块(TCB)初始化、TCB 优先级表初始化、事件控制(ECB)链表初始化以及空闲任务的创建等。

然后,根据应用程序的需要,用户可以调用  $\text{OSTaskCreate}()$  函数创建多个任务(至少 1 个)。该函数为新任务建立任务堆栈  $\text{OSTaskInit}()$  以及初始化任务控制块 TCB( $\text{OS\_TCBInit}()$ )。

最后,通过调用  $\text{PSSstart}()$  启动多任务调度。程序将调到就绪列表中优先级最高的任务开始执行。可以先假设启动多任务调度前创建了一个任务  $\text{Task1}()$ 。

程序进入  $\text{Task1}()$  后,首先初始化时钟节拍源开始计时。此节拍源给系统提供周期性的时钟中断信号,实现延时和超时确认。然后根据应用程序要求,完成该任务的基本功能。最后调用  $\text{OSTimeDly}()$ ,将自己挂起,即从就绪表中删除。只有等延时时间到,才将该任务恢复到就绪表中,并等待调度器调度。

$\text{OSTimeDly}()$  将任务挂起的同时,为其做好延时记号,然后调用  $\text{OS\_Sched}()$  进行任务级的任务调度。若此时没有任何任务进入就绪态,就切换到空闲任务。

当时钟中断来临时,系统即入中断任务程序  $\text{OSTickISR}()$ 。系统包当前正在执行的任务挂起,保护现场,进行中断处理  $\text{OSTimeTick}()$ ,判断有无空闲任务。

当时钟中断来临时,系统进入中断服务程序  $\text{OSTickISR}()$ 。系统把当前正在执行的任务挂起,保护现场,进行中断处理  $\text{OSTimeTick}()$ ,判断有无任务延时到期,若有,则使任务进入就绪态。最后调用  $\text{OSIntExit}()$  进行中断级的任务调度,从而切换到优先级最高的任务,若没有别的任务进入就绪态,则恢复现场继续执行原任务<sup>[4~6]</sup>。

## 2.2 $\mu\text{C}/\text{OS}-\text{II}$ 移植对处理器的要求

$\mu\text{C}/\text{OS}-\text{II}$  移植对处理器的要求为<sup>[4]</sup>:

- \* 处理器的 C 编译器能产生可重入代码;
- \* 在程序中可打开或关闭中断;
- \* 处理器支持中断,并能产生定时中断(通常在 10Hz~100Hz 之间);
- \* 处理器能支持容纳一定数据量的堆栈;
- \* 处理器有意将堆栈指针和其他寄存器存储到堆栈(或内存),或从堆栈(或内存)读到堆栈指针和其他寄存器的指令。

## 3 $\mu\text{C}/\text{OS}-\text{II}$ 移植开发工具及所需修改的文件

### 3.1 开发工具

实现  $\mu\text{C}/\text{OS}-\text{II}$  的移植,要求所用的 C 编译器支持

混合编程。选用 ARM ADS 编译器,仿真板基于 Samsung 公司的 S3C44B0 芯片。

### 3.2 相关文件的处理

$\mu\text{C}/\text{OS}-\text{II}$  的移植与 CPU 相关的文件主要有三个,分别是汇编文件  $\text{OS\_CPU\_A.ASM}$ 、C 语言文件  $\text{OS\_CPU\_C.C}$  和头文件  $\text{OS\_CPU\_H}$ 。

#### 3.2.1 设置 includes.h 中与处理器、编译器相关的代码

$\text{OS\_CPU\_H}$  文件中主要是定义了一些与处理器相关的常数、宏以及类型。根据 ADS1.2 编译手册,char 型变量为 8 位,short 型为 16 位,int 型为 32 位,堆栈为 32 位宽。定义了开/关中断的两个函数: $\text{OS\_ENTER\_CRITICAL}$  和  $\text{OS\_EXIT\_CRITICAL}$ 。相关代码如下:

```
# define INT8U unsigned char
# define INT16U unsigned short
# define INT32U unsigned long
# define OS_STK unsigned long
# define BOOLEAN int
# define OS_CPU_SR unsigned long /* 设置  $\mu\text{C}/\text{OS}-\text{II}$  CPU 状态寄存器数据类型 */
# define INT8S char
/* 函数声明 */

extern int INTS_OFF(void); /* 关中断 */
extern int INTS_ON(void); /* 开中断 */
# define OS_ENTER_CRITICAL()
{ CPU_SR = INTS_OFF();

# define OS_EXIT_CRITICAL()
{ if (CPU_SP == 0) INTS_ON();

# define OS_STK_GROWTH 1 /* 定义堆栈指针增长方向 */

# define STACKSIZE 286 /* 堆栈尺寸 */
INTS_OFF /* 关中断 */
/* 将状态寄存器相关位置 1 */
mrs r0,cpsr;当前状态寄存器内容送入 r0
mov r1,r0;r0 送入 r1
orr r1,r1,#0xc0;r1 中 D7、D6 置 1
msr cpsr,r1;送 CPSR,关中断
and r0,r0,#0x80;从初始 CPSR 的返回 FIQ
mov pc,lr;返回
INTS_ON /* 开中断 */
mrs r0,cpsr;取当前 CPU 的状态
bic r0,r0,#0xc0;消除 D7、D6 位
msr cpsr,r0;送到 CPSR,开中断
mov pc,lr;返回
```

在 ARM 为 16/32 为微处理器,故数据类型  $\text{OS\_STK}$  声明为 32 位。方向从高地址向低地址生长,所以结构常量  $\text{OS\_STK\_GROWTH}$  设置为 1。 $\mu\text{C}/\text{OS}-\text{II}$  在进入系统临界代码区之前要关中断,等到退出临界区后再打开,



以保护核心数据不被多任务环境下的其它任务或中断破坏。开、关中断可通过设置当前程序状态寄存器 CPSR 中的中断屏蔽位实现。

### 3.2.2 OS\_CPU\_C.C 文件与操作系统相关的函数

移植  $\mu\text{C}/\text{OS}-\text{II}$  需要在 OS\_CPU\_C.C 中定义 6 个函数,而实际上需要定义的只有 OSTaskInit() 一个函数。该函数将相关内容传给操作系统。通过调用 OSTaskInit() 来初始化任务的堆栈结构,传送任务地址、参数指针、任务的堆栈栈顶和任务的优先级给 OSTaskCreate(), 最终保存到任务控制块 OS\_TCB。

移植  $\mu\text{C}/\text{OS}-\text{II}$  在 OS\_CPU\_C.C 文件中另外 5 个函数: OSTaskCreateHook 创建扩展功能调用该函数来调用 OS\_TCB 指针; OSTaskDelHook 调用删除该函数 OS\_TCB 指针; OSTaskSwHook 调用该函数进行任务切换; OSTaskStatHook 用于统计 CPU 执行情况,用户可以扩展一个统计功能; OSTimeTickHook 称为钩子函数,可不加代码。

### 3.2.3 OS\_CPU\_A.ASM 文件与处理器相关的函数

用汇编语言编写的 OS\_CPU\_A.ASM 文件主要完成任务栈的初始化,包括 4 个与处理器相关的函数,分别是: OSStartHighRdy() 运行优先级最高的就绪任务; OS\_TASK\_SW() 用于任务级的任务切换; OSIntCtxSW() 用于中断级的切换; 以及 OSTickISR() 时钟节拍中断这 4 个函数需用固定的格式在头文件中。相关代码如下:

```
OSStartHighRdy()
LDR r4, addr_OSTCBCur; 得到当前任务 TCB 地址
LDR r5, addr_OSTCBHighRdy; 得到最高优先级任务的 TCB 地址
LDR r5, [r5]; 获得堆栈指针
LDR SP, [r5]; 转到新的堆栈中
STR r5, [r4]; 设置当前任务的 TCB 地址
LDMFD SP!, {r4}; 出栈
MSR SPSR, r4
LDMFD SP!, {r4}
MSR CPSR, r4
LDMFD SP!, {r0-r12, lr, PC}; 运行新任务
/* 任务切换 */
LDR r4, addr_OSTCBCur; 得到当前任务 TCB 地址
LDR r5, [r4]
STR SP, [r5]; 得到 SP 在被占先的任务 TCB 地址
LDR r6, addr_OSTCBHighRdy; 得到最高优先级任务的 TCB 地址
LDR r6, [r6]
LDR SP, [r6]
STR r6, [r4]; 设置新的当前任务的 TCB 地址
LDMFD SP!, {r4}; 保存任务的状态寄存器
MSR SPSR, r4
LDMFD SP!, {r4}
MSR CPSR, r4
LDMFD SP!, {r0-r12, lr, PC}
```

$\mu\text{C}/\text{OS}-\text{II}$  是可剥夺型实时多任务内核,可以剥夺实时内核在任何时刻都运行就绪了的最高优先级的任务。也就是说,高优先级的任务一旦就绪,将立刻投入运行。 $\mu\text{C}/\text{OS}-\text{II}$  调度工作的内容可分为两部分:最高优先级任务的寻找和任务切换。为了实现就绪任务的快速查找, $\mu\text{C}/\text{OS}-\text{II}$  中采用了一种非常独特的方式来标识任务就绪状态,即就绪任务表。有了就绪任务表,任务调度模块可以很方便、快速地得到最高优先级的就绪任务。

## 4 小 结

由于各方面的原因,移植中编写的代码不一定完全正确,需要进行逐步调试。调试过程中,需要根据现象来发现问题所在。

(1)对  $\mu\text{C}/\text{OS}-\text{II}$  的内核机理要有充分理解。可以尝试对其内核进行调试,这样可以帮助自己从更深的层次来理解嵌入式实时操作系统。只有对  $\mu\text{C}/\text{OS}-\text{II}$  的内核有了更清楚的认识,才能在移植过程中发现问题的本质。

(2)对所使用的编译器有更深入的了解,以节约代码编写时间。

(3)在移植过程中,最容易出问题的就是堆栈处理。堆栈处理是操作系统移植的关键部分。先分析 ARM 系统自身在进行现场保护时的堆栈处理的操作,然后模拟其过程。关键是要将需要的寄存器都保护到。由于在进行任务切换的时候采用了系统函数来进行现场保护,因此在堆栈初始化的时候,就应该按照这两个函数的操作来对任务堆栈进行初始化。

(4)要详细分析 ARM 系统自身处理中断的压栈操作,必须将多余的信息从堆栈中清理干净。在移植过程中,由于 ARM 处理器有 7 种运行模式,每种运行模式都有自己独立的寄存器,所以在处理中断过程中,要注意运行模式的切换,避免寄存器内容无法正确保存和恢复。

(5)注意程序的返回地址。异常中断发生时,程序计数器 PC 所指的位置对各种不同的异常中断是不同的,所以返回地址也是不同的。需要根据不同的中断类型,确定不同的中断返回地址偏移量,否则程序将会跑飞。

## 参考文献:

- [1] 马中梅,李善平,康 慨,等. ARM&Linux 嵌入式系统教程[M]. 北京:北京航空航天大学出版社,2004.
- [2] Krishna C M, Shin K G. Real-time systems(实时系统)[M]. 北京:清华大学出版社,2001.
- [3] 刘振鹏,李亚平,王 煜,等. 操作系统[M]. 北京:中国铁道出版社,2003.
- [4] 邵贝贝,龚光华,薛 涛,等. Motorola DSP 型 16 位单片机原理与实践[M]. 北京:北京航空航天大学出版社,2003.
- [5] 张云生. 实时控制系统软件设计原理及应用[M]. 北京:国防工业出版社,1998.
- [6] LABROSSE J J.  $\mu\text{C}/\text{OS}-\text{II}$ —源码公开的实时嵌入式操作系统[M]. 邵贝贝,译. 北京:中国电力出版社,2001.