μC/OS-II 在 ARM 平台上移植的深入探讨

王琨强，赵志珩
（汽车管理学院 基础部，蚌埠 233011）

摘　要：在以 S3C2410 处理器的嵌入式平台上，把经典的 vivi 启动代码与 μC/OS-II 操作系统结合在一起，探讨了 μC/OS-II 的移植实现，尤其详述了在 ARM 处理器 ISR 中断模式下如何实现断点数据保护的方法。利用该方法，可以将一般 ARM 系统的启动代码同 μC/OS-II 操作系统融合起来，对于 μC/OS-II 操作系统在 ARM 平台的推广和 μC/OS-II 操作系统的研究都很有意义。

关键词：嵌入式操作系统；μC/OS-II；ARM；S3C2410

中图分类号：TP316.2　　**文献标识码**：A　　**文章编号**：1674-7720(2010)12-0007-04

The transplant analysis and research of μC/OS-II based on ARM system

WANG Kun Qiang，ZHAO Zhi Heng
(Dept. of Basic Courses, Automobile Management Institute, Bengbu 233011, China)

Abstract: Based on the platform of S3C2410, the method of combining its bootloader, vivi, and the embedded operating system, μC/OS-II, is introduced in this paper, and the transplant key points, realization approaches are also given. The transplant has been inspected and verified. The analysis in this paper is useful for the application of μC/OS-II in ARM system.

Key words: embedded operating system; μC/OS-II; ARM; S3C2410

μC/OS-II 在 ARM 平台的移植是一个重要的学习过程，有助于提高对 RTOS 的认识与理解，从而提高嵌入式工作者的理论与技术水平。μC/OS-II 是一个小的实时内核，源代码公开，有详尽的解释。正是因为其内核小，才便于研究、理解和掌握。另外，参照 TCP/IP 协议、标准和一些公开的图书，在 μC/OS-II 上增加 TCP/IP 协议栈，蓝牙通信软件、红外通信协议也十分方便，商业价值得到了认可。

随着科技的发展，嵌入式应用的复杂性越来越高，同时 ARM 体系处理器的价格越来越低，ARM 平台 + 实时操作系统的架构体系的使用会越来越广泛。有鉴于此，本文对 μC/OS-II 在 ARM 平台下的移植进行了深入探讨。

1 操作系统 μC/OS-II 及 S3C2410 开发平台简介

1.1 μC/OS-II 简介

μC/OS 最早出自于 1992 年美国嵌入式系统专家 Jean J.Labrosse 在《嵌入式系统编程》杂志 5 月和 6 月上刊登的文章连载，并把 μC/OS 的源代码发表在该杂志的 BBS 上。μC/OS-II 是目前最新的版本。

μC/OS-II 是专门为计算机的嵌入式应用而设计的，绝大部分代码用 C 语言编写。CPU 的相关部分采用汇编语言编写，总量在 200 行左右的汇编语言被压缩到最低限度，目的是便于移植到任何一种其他的 CPU 上去。μC/OS-II 具有执行效率高、占用空间小、实时性优良、可扩展等特点，最小内核可编译至 2 KB。μC/OS-II 可移植到几乎所有知名的 CPU 上。

1.2 μC/OS-II 的组成

严格地说 μC/OS-II 只是一个实时操作系统内核，它仅仅包含了任务调度、任务管理、时间管理、内存管理和任务间的通信和同步等基本功能。没有提出输入输出管理、文件系统、网络通信等额外的服务。但由于 μC/OS-II 良好的可扩展性和源代码开放，这些非必须的功能完全可以由用户根据自己的需要分别实现。

μC/OS-II 可以大致分成核心、任务处理、时间处理、任务同步与通信、CPU 的移植等 5 个部分[1]。

(1)核心部分(OSCore.c)：操作系统的处理核心，包括操作系统的初始化、操作系统运行、中断进出的前导、时钟节拍、任务调度、事件处理等多部分。

(2)任务处理部分(OSTask.c)：与任务操作密切相关的部分。包括任务的建立、删除、挂起、恢复等等。

(3)时钟部分(OSTime.c)：μC/OS-II 中最小的时钟单位是 timetick(时钟节拍)。任务延时等操作在此完成。

(4)任务同步和通信部分:为事件处理部分,包括信号量、邮箱、邮箱队列、事件标志等部分,主要用于任务间的相互联系和对临界资源的访问。

(5)与 CPU 的接口部分:这里是指 μC/OS-II 针对所使用的 CPU 需要改写的部分。由于 μC/OS-II 是一个通用性的操作系统,其开放的源代码是以 X86 内核为例而编写的,在应用到其他处理器平台上时,这部分代码必须做相应的改变。

1.3 ARM 硬件开发平台简介

调试时所用的硬件开发平台是一款基于三星 S3C2410A 芯片的开发平台。S3C2410 开发板是一款通用的 ARM9 开发板,其基本配置采用三星公司的 S3C2410 ARM920T 芯片,主频 203 MHz。集成有 SDRAM 控制器、NAND Flash 控制器、SD 读卡器、USB Host 和 USB Device 控制器、LCD 控制器、I²C 总线控制器、SPI 总线接口等。开发板上 Flash 空间为 32 MB,SDRAM 容量为 128 MB。

2 S3C2410 引导程序

开发板原有引导程序由 VIVI 公司提供,其运行过程分成两个阶段。第一阶段的代码用汇编语言编程,主要完成以下任务:(1)初始化 CPU 速度、存储器、存储器配置寄存器,以及串口等硬件资源的配置;(2)建立内存空间的映射图,将系统的软硬件环境带到合适的状态,为最终调用操作系统内核做准备;(3)装载操作系统映像到内存中;(4)设置相关寄存器和资源,跳转到 main() 函数,进入第二阶段。

第二阶段的代码用 C 语言编写,从 main() 函数开始,主要工作有:开发板外部接口初始化(I/O 接口、UART 接口、LCD 接口等)、内存映射和内存管理单元初始化等,最后启动 linux 内核。有大量文章对此开发板引导程序作了详细的分析[3],本文在这里不做重复,本文的重点是将引导程序与 μC/OS-II 操作系统二者融合,既利用了开发板源代码提供的关于 UART 口、LCD 和触摸屏接口程序;时钟、内存管理等丰富的驱动程序和接口程序,又成功地完成了对 μC/OS-II 实时操作系统的移植和整合。

3 移植要点

μC/OS-II 的内核分成 2 个部分,与处理器无关的代码和与处理器有关的代码。移植过程中需要根据 S3C2410 处理器和 ADSV1.2 开发平台(这里特地强调编译平台的因素,主要考虑到各个编译平台对数据格式的理解略有差别)的特点来重新编写 3 个文件,用 C 语言编写的 OS_CPU.H、OS_CPU_C.C 和用汇编语言编写的 OS_CPU_A.ASM,此外,要将 S3C2410 开发板引导程序和 μC/OS-II 内核程序融合在一起,还必须将各自 main() 函数融为一体。

3.1 OS_CPU.H 的移植

μC/OS-II 内核中 OS_CPU.H 代码是根据 X86 内核而写的,其中的数据格式定义与 ARM9 内核以及 ADSv1.2 开发平台不完全相符。OS_CPU.H 的移植分为以下 4 个部分:

(1)数据类型定义:在调试时发现,虽然定义 8 bit 或 16 bit 数据类型时,在编译过程中不会报错,但这些变量并不会按要求被正确初始化或赋值,运行过程常常出错。所以,在改写 OS_CPU.H 代码时,将所有变量都定义成 32 bit 或 64 bit;

(2)堆栈生长方向定义:ARM 的堆栈是从上往下生长的,OS_STK_GROWTH 定义为 1;

(3)开关中断的宏定义:用开关中断的汇编函数实现,放在 OS_CPU_A.ASM 文件中。

(4)宏定义 OS_TASK_SW():这个宏定义是在 ARM 中断处理之外时,μC/OS-II 从低优先级切换到高优先级任务时所调用的代码,它总是在任务级代码中被调用。在有些资料中[1],将 OS_TASK_SW() 和 OSIntCtxSw() 等同起来,这在 ARM 内核中是不行的,因为后者是 ARM 内核在中断模式下的任务切换函数,而不同模式下处理器的寄存器组是不同的,所要保护的寄存器内容也不相同,经过调试,发现以下代码可达到目的。

```
OS_TASK_SW
    stmfd   sp!, {lr}; PC 入栈,lr 其实是任务的返回地址,
    stmfd   sp!, {r0-r12, lr}
    mrs     r4, cpsr
    stmfd   sp!, {r4}                   ;最后保存 CPSR
    ldr     r4, =OSTCBCur
    ldr     r5, [r4]
    str     sp, [r5]    ;将 SP 保存在当前任务的控制块中
    ldr     r5, =OSTCBHighRdy
    ldr     r5, [r5]
    str     r5, [r4]            ;OSTCBCur = OSTCBHighRdy
    ldr     r6, =OSPrioHighRdy
    ldr     r6, [r6]
    ldr     r4, =OSPrioCur
    str     r6, [r4]            ;OSPrioCur = OSPrioHighRdy
    ldr     sp, [r5]              ;得到新任务的堆栈指针
    ldr     r4, [sp], #4
    msr     cpsr_cxsf, r4                   ;先恢复 CPSR
    ldmfd   sp!, {r0-r12, lr, pc}
```

3.2 OS_CPU_C.C.H 的移植

在 OS_CPU_C.C 中,最主要的函数是 OSTaskStkInit(),它在任务建立时,用来初始化任务堆栈结构,其余钩子函数可以不用动,这个函数的代码比较简单[2]。需要说明的是,由于本文所述系统,用户任务运行在 SVC 模式下,没有保存 SPSR 寄存器。

3.3 OS_CPU_A.ASM 的移植

OS_CPU_A.ASM 文件的汇编程序是 μC/OS-II 移植工程的重点和难点。它通常包括 OSStartHighRdy()、OS-

IntCtxSw()、OSTickISR()和开关中断代码等。其中,OSStartHighRdy()的主要工作是将优先级最高任务对应的所有寄存器按顺序从任务堆栈中恢复出来,其代码简单[2]。对于开关中断函数,在调试时所用代码如下:

```
EnterCritical
    mrs  r1, cpsr
    str  r1, [r0]
    orr  r1, r1, #NOINT
    msr  cpsr_cxsf, r1
    mov  pc, lr
ExitCritical
    ldr  r1, [r0]
    msr  cpsr_cxsf, r1
    mov  pc, lr
```

需要指出的是,在每次成对调用这两个函数时,需要提前声明变量 r,代码如下所示:

```
INT32U          r;
EnterCritical(&r);
ExitCritical(&r);
```

需要慎重对待的是 OSIntCtxSw()、OSTickISR()函数。在调试时发现,用一般参考资料所介绍的代码都无法实现多任务的正常运行,其主要原因是,对 ARM9 内核而言,其每种特定的中断返回,都有特定的返回指令,在中断处理过程中,强制使用模式切换指令,使处理器的中断处理机制发生混乱,程序无法正常执行。例如在 ISR 模式中使用指令:

```
MSR CPSR_c, #(NO_INT | SVC32_MODE)
```

其目的是返回 ISR 发生之前的模式,然后保存一些寄存器。但调试时发现,在上述指令执行之后,处理器重新响应 ISR 中断,并没有顺序执行,而是立即回到 ISR 模式下。

还有,对于 S3C2410 的 ARM920T 内核而言,其 ISR 模式的返回指令是:

```
ldmfd   sp!, {r0-r12, lr}
subs    pc, lr, #04
```

其他任何形式的指令都无法使处理器正确返回。有些资料用下述指令:

```
Ldmfd  sp!, {r0-r12, lr, pc};
```

执行之前堆栈中相应存储单元的内容为(lr-4)。

看起来与前面的两行代码意义相同,但后面的代码仅仅让处理器实现 PC 指针的跳转,而无法实现处理器的模式转换,即从 ISR 模式回到中断发生之前的模式。

但在中断发生时,无法在中断处理过程中保存所有的处理器寄存器。例如,在 ISR 模式下,无法保存 SVC 模式的 LR 寄存器等。为了解决这个问题,本文采取了如图 1 所示的框图结构来编写中断处理代码和 OSIntCtxSw()函数。

因为 S3C2410 在进入 ISR 模式后,自动屏蔽 ISR 中断,所以程序在中断嵌套,可以表明 2 个全部变量 ISR_LR 和 ISR_SPSR 用于保存 ISR 中断发生之时处理器的 lr 和 spsr 寄存器。其代码的特别之处在于,在 ISR 中断处理过程中通过修改 lr 寄存器,而使处理器在退出 ISR 模式时能根据任务的需要返回至 ISR 中断发生之处或者代码指定地点。在代码指定地点,可以保存上次中断发生时被中断任务的处理器的所有寄存器数据。这里需要注意一点,当处理器退出 ISR 模式时跳转到 Saveregister 处开始执行命令,需要提前将 Saveregister 处的地址加上 4,然后赋值给 lr 寄存器。因为在 ISR 退出时,需要将 lr 减去 4 再赋值给程序计数器 pc。

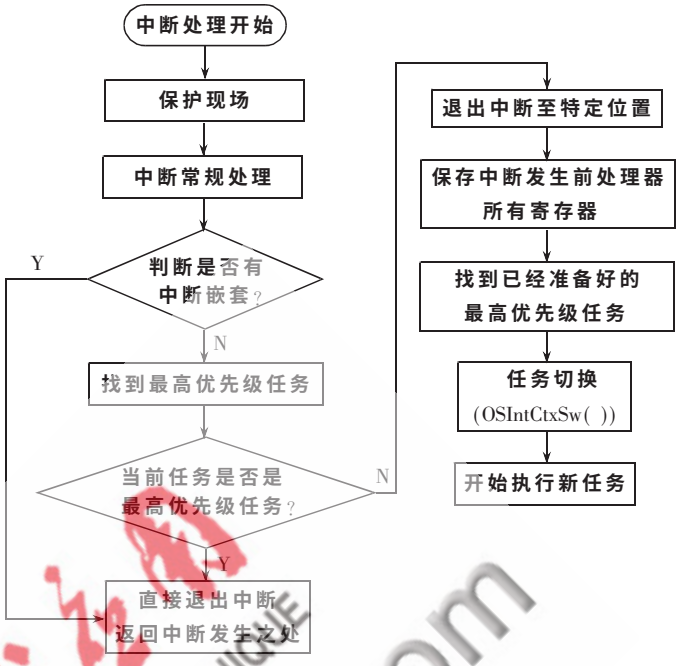

图 1 中断处理过程

4 S3C2410 启动代码和 μC/OS-II 的融合

本文 1.1 节已经介绍过,S3C2410 的启动代码开始部分是汇编语言的初始化过程,然后跳转到 main()函数。融合的工作就从改造 S3C2410 的 main()函数和 μC/OS-II 的 main()函数(在 test.c 中)开始。在 S3C2410 的 main()函数中,保留原启动代码中关于端口、内存、外部设备初始化代码,删去跳转到 Linux 操作系统的代码;在 μC/OS-II 的 test.c 文件的 main()函数中,删去一切与 X86 内核有关的初始化代码和输入输出函数代码(因为这部分代码在 S3C2410 的启动代码中已经实现),并将与 μC/OS-II 内核有关的 3 个函数 OSInit()、OSTaskCreate(…)、OSStart()复制到 S3C2410 的 main()函数中,同时删去 μC/OS-II 的 test.c 文件。融合后的 main()函数主要代码如下:

```
ChangeClockDivider(1, 1);                    //1:2:4
ChangeMPllValue(161,3,1);                    //FCLK=203.0 MHz
SetClockDivider(1, 1);
SetSysFclk(FCLK_203M);
```

```
Port_Init();
Isr_Init();
Uart_Init(0, 115200);
Uart_Select(0);
MMU_Init();                                    //MMU 初始化
EnableModuleClock(CLOCK_ALL);
rMISCCR &= ~(0x3007);
OSInit();
OSTaskCreate(TaskStart,……, 0);
OSStart();
```

至此,处理器已执行完 S3C2410 的启动代码,并开始执行 μC/OS-II 内核代码。当然,要实现多任务,处理器的中断必须是打开的。这个工作在 OSStart ()函数中完成,在执行 OSStartHighRdy 之前,要按照系统的需求完成处理器的中断初始化工作,同时打开中断。至此,融合工作基本完成,剩下的工作就是按照系统的需求在 μC/OS-II 的 TaskStart(…)函数中自由添加实际工作所需的任务了。

在本文所述系统中,在 μC/OS-II 所带 3 个系统任务的基础上添加了 3 个任务 Task1、Task2 和 Task3,方法是在 OSStatInit( )之前添加 OSTaskCreate(Task1,…)等代码,然后按下述格式和自己的需求编写 Task1、Task2 和 Task3 函数。代码为:

```
void Task1(void *data)
{
    while(1) { ;任务代码 }
}
```

因篇幅所限,无法详述在融合过程中遇到的所有问题,尤其是在 ADSv1.2 环境下编译、调试过程出现的语法问题和各种细节问题。

随着科技的发展和实际任务复杂性的逐步增加,传统的单片机前后台编程模式渐渐不能满足实际应用的要求。在嵌入式应用开发中使用嵌入式操作系统已经成为一种趋势,本文在 S3C2410 开发板上将原有的引导程序和 μC/OS-II 操作系统结合在一起,开发出能自引导的 μC/OS-II 操作系统,该系统除了 3 个系统任务外,还自带 3 个实际任务,在 ADSV1.2 环境下编译、调试,并在板卡上成功运行,对 μC/OS-II 在 ARM 平台上的移植有一定借鉴意义。

参考文献

[1] 任哲,潘树林,房红征.编著.嵌入式操作系统基础 μC/OS-II 和 Linux[M].北京:北京航空航天大学出版社,2007.

[2] 韩山,郭云,付海艳,编著.ARM 微处理器应用开发技术详解与实例分析[M].北京:清华大学出版社,2007:284-286.

[3] 蒋维. 基于 ARM S3C2410 嵌入式系统的 Bootloader 分析与设计[J].电子工程师,2008(10).

(收稿日期:2010-02-25)

作者简介:

王琨强,男,1968 年生,讲师,硕士,主要研究方向:嵌入式操作系统,信号与信息处理。

赵志珩,男,1966 年生,副教授,硕士,主要研究方向:嵌入式操作系统。