提高μC/OS-II在ARM上执行效率的几种方法

李章林[1] 卢桂章[1] 辛运帏[2]

(1.南开大学信息技术科学学院机器人所，天津 300071; 2. 南开大学信息技术科学学院计算机系，天津 300071)

摘要： μC/OS-II的执行效率主要取决于任务切换时间，而任务切换时，需进行堆栈操作。ARM处理器有多种运行模式，每种运行模式有不同的堆栈，这使得在ARM处理器上实现任务切换有其特殊性，本文利用ARM处理器在任务切换时堆栈的变化特点，优化设计堆栈操作方式和运行模式转化方式，以减小任务切换时间。实验结果表明该方法比其它实现方法具有更高的实时性。同时，本文采用在ARM上实现可重入中断来减少高优先级任务切换时间。

关键词： μC/OS-II; ARM; 实时性;

Some Methods for Promoting Execution Efficiency of μC/OS-II on ARM Platform

Li Zhanglin, Lu GuiZhang, Xin Yunwei

(Information Technology Department, Nankai University, Tianjin 300071)

Abstract: The executing efficiency of μC/OS-II mainly depends on task swap time, and it needs operate stack during task swap. ARM MCU has multi-executing-mode, and there is distinct stack for each mode, so there is particularity in implementing task swap on ARM platform. The paper optimized the stack operation pattern and executing-mode transform pattern according to the characteristic of stack operation during task swap on ARM platform, in order to reduce the task swap time. The experiment result showed that this method had advantage over the others. Furthermore, the paper realized the reentrant interruption on ARM platform, in order to reduce the task swap time for high priority tasks.

Keywords: μC/OS-II; ARM; Real-time;

1 引言

μC/OS-II 是一个公开源码的抢占式、多任务的实时操作系统，因其具有开源性、实时性强、代码紧凑、稳定可靠等特点在各种系统中得到了广泛应用。μC/OS-II 在 ARM 处理器上的移植也已经实现[1,2,5]。ARM 处理器加μC/OS-II 操作系统的嵌入式系统常用于工业实时控制，对执行效率特别是实时性要求较高。文献[1]提出了减少代码量的方法，即将需要移植的汇编函数的公共部分提取出来写成子函数的形式；文献[2] 采用 scatter-loading 和让系统从内部 RAM 里读取异常向量表相结合的机制把系统调用频繁且对系统执行效率及实时性影响较大的程序放到内部 RAM 里执行的解决方案。

虽然这些方法都是提高执行效率的有效方法，但是它们都没有考虑到 ARM 处理器本身的特点。提高μC/OS-II 执行效率的关键是减少任务切换时间，任务切换时主要的操作是任务栈的出栈和入栈操作。由于 ARM 处理器在不同的处理器运行模式具有不同的堆栈，合理利用该特性并结合处理器运行模式切换特点能够减少任务栈操作次数，减少任务切换时间。本文主要提出了两类提高μC/OS-II 在 ARM 上执行效率的方法：①减少任务栈操作次数。②ARM 处理器上的μC/OS-II 的可重入中断的实现

2 μC/OS-II在ARM上通常的任务切换方式

μC/OS-II 中进行任务切换的函数是 OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()。其中 OSStartHighRdy()是在操作系统第一次启动的时候调用的，OSCtxSw()是用户主动调用进行任务切换，这两个函数和中断无关，所以和 ARM 的处理器模式无关，在 ARM 上的实现方法和一般处理器类似。

OSIntCtxSw()用于在中断中进行任务切换。嵌入式实时系统响应外部事件主要通过外部中断，在中断中将高优先级的任务设置为就绪状态，然后在中断退出时自动切换到高优先级的任务。为了实现该功能，μC/OS-II 要求用户在中断中：

（1） 若需运行高优先级任务，设置该高优先级任务的就绪标志位。

（2） 在中断函数末尾调用 OSIntExit()函数，OSIntExit()函数检查是否有高优先级任务

天津市应用基础研究计划项目支持,项目编号:06YFJMJC00200
天津市高等学校科技发展基金项目支持,项目编号:20051518
南开大学创新基金支持

就绪，有则调用 OSIntCtxSw()，并切换到该高优先级任务。

进行任务切换时需将当前任务的状态变量压入堆栈（也就是当前任务栈），然后将堆栈指针指向高优先级任务的任务栈，并弹出该高优先级任务堆栈中的状态变量。由于 OSIntCtxSw()在中断中调用，而进入中断函数后一般已经保存状态变量到堆栈，于是 OSIntCtxSw()不需要进行上述压栈操作。但是由于 ARM 体系结构决定了中断和非中断中使用不同的运行模式和堆栈，使得该问题变得复杂。例如，假设在非中断中使用 SVC 模式（管理模式）[4] 和 SVC 堆栈；中断中使用 IRQ 模式（外部中断模式）和 IRQ 堆栈；任务切换时需要保存的状态变量为 PC,R12-R0, RL,CPSR,SPSR，且它们任务栈中的排列顺序如图 1 所示。设中断函数为 OSInt1()，则通常在 ARM 上实现 OSInt1()和 OSIntCtxSw()可用表 1、表 2 的伪代码表述：

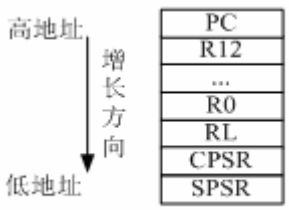

图1.任务栈结构图

表 1. OSInt1 中断服务程序的伪代码

OSInt1
{
将 LR_irq-4、R12 至 R0 的内容依次入栈; /*LR-4 是中断服务程序的返回地址即 PC*/ **/* A 处 */**
清除中断标志位; **/* B 处 */**
调用 OSIntEnter;
中断事件处理，例如使得高优先级任务就绪;
调用 OSIntExit;
将堆栈的内容依次弹出到 R0 至 R12、PC 寄存器中，同时从 SPSR_irq 恢复状态寄存器;
}

表 2. OSIntCtxSw 程序的伪代码

OSIntCtxSw
{
调整堆栈指针 SP 到表 1 中"A 处"; **/* C 处 */**
保存 SPSR_irq(中断返回状态寄存器)，从堆栈中弹出 R0-R12, LR_irq，并保存 LR_irq（中断返回地址）; **/* D 处 */**
强制切换到 SVC 模式;
将 LR_irq, LR_svc, R12-R0, CPSR_irq, SPSR_irq 依次压入堆栈; **/* E 处 */**
调用_OSCtxSw;/* _OSCtxSw 是 OSCtxSW 函数去除入栈操作的部分 */
}

从表 1、表 2 可见在 ARM 处理器上 OSIntCtxSw()不仅没有节省压栈操作，而且还多了一步出栈操作。这是由于中断函数 OSInt1()压入的状态变量不在任务栈（即 SVC 堆栈）中而是在 IRQ 堆栈中，所以 OSIntCtxSw()必须重新从 IRQ 堆栈中弹出状态变量，并将其压入 SVC 堆栈。该额外的出栈入栈操作增加了任务切换时间。这也是现有的μC/OS-II 在 ARM 上实现的方式[1,2,5]。

3 减少任务栈操作次数的改进

为了防止 OSIntCtxSw()中进行额外的出栈入栈操作，本文提出了两种改进方案：

3.1 中断中使用 SVC 堆栈

假如在中断中直接使用 SVC 堆栈，那么进入中断函数后的压栈操作将状态变量直接压入了任务栈，这样就不需要在 OSIntCtxSw()中进行额外的出栈和入栈操作。实现方法是，在进入中断函数的第一时刻，将 SVC 堆栈寄存器 SP_svc 的内容拷贝到 IRQ 堆栈寄存器 SP_irq 中。此后虽然中断中仍然使用 SP_irq 寄存器，然而实际指向任务栈。本文实现该方案的伪代码如表 3、表 4 所示：

表 3. 改进 1 的 OSInt1 中断服务程序的伪代码

OSInt1
{
切换到 SVC 模式;
将 SP_svc 保存到临时变量 save_sp_svc 中;
切换回 IRQ 模式;
将 save_sp_svc 的内容加载到 SP_irq 中;
跳转到_OSInt1;/* 这里的_OSInt1 和表 1 的 OSInt1 完全一样 */
}

表 4. 改进 1 的 OSIntCtxSw 程序的伪代码

OSIntCtxSw
{
调整堆栈指针 SP_irq;
保存 SPSR_irq(中断返回状态寄存器)到 R0，保存 SP_irq 到 R2;
强制切换到 SVC 模式;

```
    复制 R2(=SP_irq) 到 SP_svc; 将 LR_svc，
    R0(=SPSR_irq),R0(=CPSR),依次压入堆栈;
    调用_OSCtxSw;
}
```

本文实际汇编程序实现表 1、2 的代码共需 45 次出栈入栈操作，而实现表 3、4 的代码减少到 17 次出栈入栈操作。

3.2 中断中使用 SVC 模式

§3.1 的改进需要首先切换到 SVC 模式，然后切换回 IRQ 模式（如表 3 所示），最后在 OSIntCtxSw（）中又要切换到 SVC 模式（如表 4 所示），假如在 IRQ 中直接使用 SVC 模式，即可以同样使用 SVC 堆栈达到§3.1 改进的效果，同时减少了在 SVC 和 IRQ 之间来回切换的麻烦。本文实现在中断中使用 SVC 模式的伪代码如表 5、6 所示：

表 5．改进 2 的 OSInt1 中断服务程序的伪代码

```
OSInt1
{
    保存 lr-4 到临时变量 INT_RET_ADDR;
    将 SPSR_irq（返回后的 CPSR 内容）保存到
    INT_RET_CPSR;
    强制切换到 SVC 模式;
    将 INT_RET_ADDR,R12-0,LR_svc, INT_RET_CPSR
    依次入栈;
    清除中断标志位;                    /* F 处 */
    调用 OSIntEnter;
    中断事件处理，例如使得高优先级任务就绪;
    调用 OSIntExit;
    从堆栈中弹出 INT_RET_CPSR，写入 CPSR;
    依次弹出 LR_svc, R0-12, PC;
}
```

表 6．改进 2 的 OSIntCtxSw 程序的伪代码

```
OSIntCtxSw
{
    调整堆栈指针 SP;
    将 SPSP_svc 压入堆栈;
    调用_OSCtxSw;
}
```

实际代码中该方案的出栈入栈次数也是 17 次。但该方法要求在 IRQ 模式切换到 SVC 模式之前保存 IRQ 模式下的一些变量（如 SPSR_irq、LR_irq）这增加额外的一些操作。§3.3 的实验表明该改进的效率没有§3.1 的改进高效，但是该改进为§4 的可重入中断做好了准备。

3.3 任务切换时间的比较

为验证改进方法对减少任务切换时间的效果，本文设计如下实验：

让μC/OS-II 的一个任务 Task1 等待（用 OSSemPend 函数）信号量 mysem，此时 Task1 挂起。当键盘按下时进入 OSInt1 中断函数，在该中断中发送（使用 OSSemPost 函数）mysem 信号量，使得 Task1 进入就绪状态。当 OSInt1 退出时切换到 Task1 任务。

对于三种实现方法：通常的任务切换方法、中断中使用 SVC 堆栈的任务切换方法和中断中使用 SVC 模式的任务切换方法，分别测试键盘按下到切换到 Task1 的时间差。为此在 OSInt1()中断函数开始处获取定时器初始值 t1，在进入 Task1 以后读取定时器当前值 t2，则 t=|t2-t1|就是想要的时间差。

对三种实现方法分别进行 43 次实验，测得的 t 如图 2 所示。实验时，定时器以 PCLK/2（PCLK 设置为 50700000）的速率减法计数。

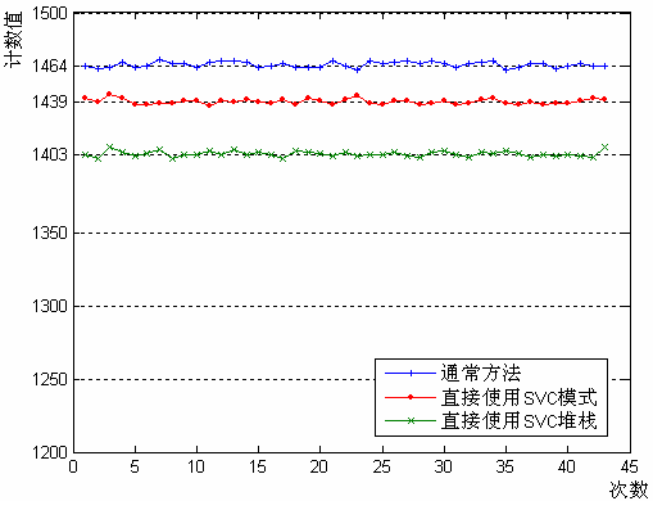

图 2．三种方法的任务切换时间比较

从图 2 可知，两种改进的方法比通常方法需要更少的任务切换时间；§3.2 的方法比§3.1 的方法需要更多的任务切换时间与原先的分析符合。

从改善的性能上，这里可以从计数值的相对值来看，§3.2 的方法比通常方法约改善了 61/1403=4.3%，但是必须考虑到时间 t 包含了中断产生到切换到 Task1 的整个代码的执行时间，其中包含了 OSSemPost()、OSIntExit()等较复杂函数，所以可以说该改进是有效的。

4 可重入中断对任务切换时间的影响

4.1 μC/OS-II 对可重入中断的支持

μC/OS-II 本身支持可重入中断[3]，它在 OSIntExit 中使用 IntNesting 变量表示了中断嵌套的层数，μC/OS-II 只在退出最后一层嵌套时才进行任务的切换，这可以通过检查 IntNesting 变量来获知。

中断的嵌套增加了任务切换的实时性，以例子说明如下：假设有两个任务：taskTimer 和 taskKey，taskTimer 由定时器触发；taskKey 由外部中断源（例如按键）触发，taskKey 的优先级比 taskTimer 高。如图 3 所示，当外部中断发生时，若处理器正好处于定时中断中。中断可重入时，系统马上进入外部中断服务程序让 TaskKey 就绪，并在退出最后一层中断（定时中断）时，切换到 TaskKey；中断不可重入时，需要首先切换到低优先级任务 TaskTimer，然后再次进入中断，然后才切换到高优先级任务 TaskKey。

由此可见中断可重入时，在有多个优先级的任务共存的情况下，高优先级任务的实时响应时间更短了。

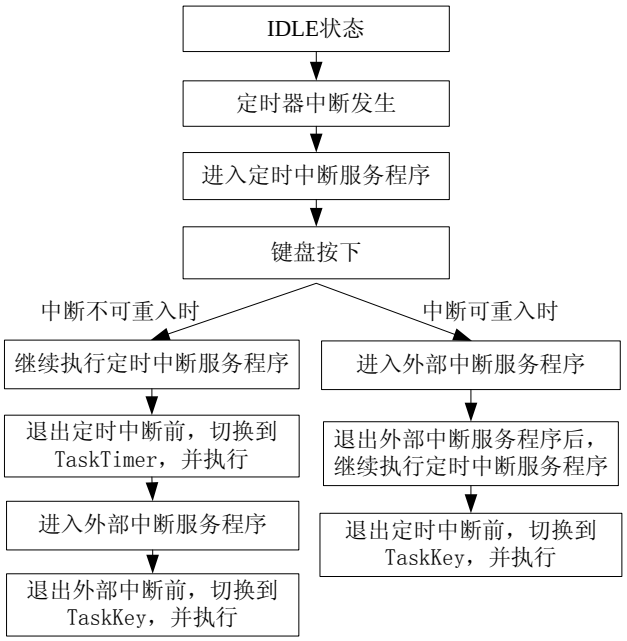

图3.中断可重入和不可重入时任务切换的差别

4.2 ARM 处理器对可重入中断的支持

在默认情况下 ARM 处理器不支持可重入中断。当 ARM 进入 IRQ 中断以后自动关闭 IRQ 使能标志，禁止 IRQ 中断的嵌套。为了支持 IRQ 的可重入，需要[6]：

1. 在重新使能 IRQ 中断之前保存必要的寄存器。
2. 在重新使能 IRQ 之前使用 SVC 模式，并在中断退出以前一直保持 SVC 模式。

4.3 在 ARM 上实现μC/OS-II 可重入中断

§3.2 的方法实际已具备可重入中断的要求，只要在表 5 的“F 处”之后重新使能 IRQ 中断标志位即可。在重新使能 IRQ 中断之前需要注意的是首先需要清除中断源的中断位，防止重新使能 IRQ 中断标志位后同一中断源重复中断。

5 结论

本文提出了两点提高μC/OS-II 在 ARM 上执行效率的方法：减少任务栈操作次数和实现可重入中断。在中断中直接使用SVC堆栈和在中断中直接使用SVC模式的改进减少了额外的出栈入栈操作，减少了任务切换时间，实验结果证实该改进的有效性。可重入中断在 ARM 处理器的μC/OS-II 上的实现，使得高优先级任务的实时响应时间更短了。两点提高效率的方法不仅有助于提高 ARM 上μC/OS-II 的执行效率。同时也对在 ARM 上实现高实时的嵌入式操作系统有借鉴意义。

参考文献：

[1]. 杨洪亮,胡金演. μC/OS-II 在 ARM 处理器上的移植[J].微计算机信息,2005,第 21 卷第 2 期:102

[2]. 王庆祎,陈曦,刘鲁源. 提高移植了μC/OS-II 的 ARM 嵌入式系统执行效率和实时性[J]. 计算机工程与应用,2005,第 22 卷:94-95

[3]. Labrosse J J. 邵贝贝译.嵌入式实时操作系统μC/OS-II.北京:北京航空航天大学出版社,2003-5.

[4]. 李明. μC/OS-II 在 ARM 上的移植[J].电子设计应用,2003,第 4 卷:p47

[5]. 张道德,杨光友,苏旭武等. 基于 ARM 架构移植μC/OS-Ⅱ的任务调度[J].湖北工业大学学报. 2005-6,第 20 卷第 3 期:12

[6]. 费浙平. 基于 ARM 的嵌入式系统程序开发要点（四）[J]. 单片机与嵌入式系统应用, 2003,11:84-85