嵌入式操作系统中断现场保护的优化策略

An Optimizing Strategy of Interrupt Context Saving in Embedded Operating System

(同济大学) **钱贾敏 王力生**

Qian,Jiamin Wang,Lisheng

摘要：大多数嵌入式操作系统在进入中断后进行中断现场保护时选择保护所有通用寄存器，这种普遍的做法延长了微处理器的访存时间，并且增加了产生存储器分体冲突的可能性。本文提出根据中断服务子程序对通用寄存器的需求而进行中断现场保护的策略，有效减少了中断现场保护时所需保护的通用寄存器数量，缩短了中断响应时间，提高了系统的中断实时性。最后，对这种优化策略进行了总结。

关键词：嵌入式系统；实时性；中断响应；中断延迟；现场保护

中图分类号：TP316.2 文献标识码：A

文章编号：1008-0570(2004)011-0083-02

Abstract: Most embedded operating systems tend to save all the processor general-purpose registers when interrupt arrives to prevent the context destroyed by interrupt service routine. As a matter of fact, in such way it delays the memory access time and rises the possibility of collision in memory banks. Thereby, an optimizing strategy is proposed in this paper that not all the general-purpose registers should be saved but only the very part that are used in the interrupt service routine. In this way, the amount of registers to be saved is decreased so that the interrupt response time is shortened and the interrupt real-time is promoted.

Keywords: embedded system; real-time; interrupt response; interrupt latency; context saving

1 嵌入式系统的实时性

嵌入式系统是以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统，而高实时性是嵌入式系统的基本要求。

IEEE (美国电气电子工程师协会)定义实时系统为 "那些正确性不仅取决于计算的逻辑结果，也取决于产生结果所花费的时间的系统"。实时系统一般可分为硬件实时和软件实时这两大类：硬实时系统有一个强制性的、不可改变的时间限制，它不允许任何超出时限的错误。超时错误会带来损害甚至导致系统失效、或者系统不能实现它的预期目标。软实时系统的时限是柔性灵活的，它可以容忍偶然的超时错误。失败造成的后果并不严重，仅仅是轻微的降低了系统的吞吐量。

2 中断响应时间

中断的实时性是实时系统的一个重要方面。中断响应时间是影响中断实时性的主要因素。中断响应定义为从中断发生到开始执行用户的中断服务代码来处理这个中断的时间[1]，其中包括中断延迟时间和保护中断现场的时间。所有实时系统在进入临界区代码段之前都要关中断，执行完临界代码之后再开中断。中断延迟时间即是从发出中断请求到任务开中断的这段时间[1]。保护中断现场有两个作用。首先是为了保护中断前任务的现场。其次，如果发生中断嵌套，还必须保护上层中断的现场。因此，整个中断响应过程如图 1 所示。要让中断服务尽快得到处理，就必须减少中断响应时间。但是从图中可以看出，中断延迟时间是由中断前任务决定的，在进入中断时只能通过尽量缩短中断现场保护的时间来达到减少中断响应时间，从而提高中断实时性。

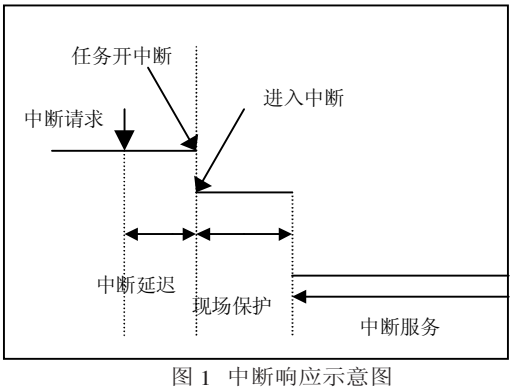

图 1 中断响应示意图

3 中断现场保护的改进

3.1 传统中断现场保护方法

对于现在大多数嵌入式操作系统，在进入中断时首先做的第一件事就是保护中断发生前的现场，即保存返回地址、程序状态字、堆栈指针以及所有通用寄存器到中断堆栈，以防止用户中断服务子程序对中断返回后现场的破坏。以 μC/OS-II 微内核为例，在 ARM 和 X86 两种体系结构微处理器上进入中断后保存现场的过程如图 2 所示。从代码中可见，两种不同的体系结构中，为保护现场，都需要执行三条访存指令，其中一条为批量访存指令 (STMFD SP! ,{R0-R12}和 PUSHA)用以保护通用寄存器 R0-R12 和 AX，CX，DX，BX，SP，BP，SI，DI。

```
ARM:
        STMFD     SP!,{LR}
        STMFD     SP!,{R0-R12}
        MRS       R0,CPSR_c
        STMFD     SP!,{R0}

X86:
        PUSHA
        PUSH      ES
        PUSH      DS
```

图 2 ARM、X86 上 μC/OS-II 中断现场保护

钱贾敏：硕士研究生

电话：010-62132436，62192616(T/F)

根据量化公式[2]：

$$\text{CPU time} = \text{IC} \times \text{CPIexecution} \times \text{Clock cycle time} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty} \times \text{Clock cycle time}$$

公式中以 CPU 时间来衡量微处理器体系结构的性能。其中前半部分是指令的执行时间，包括取指、分析、执行等，而后半部分表明如果指令是访存指令则在 cache 不命中时 CPU 时间还应该加上访存的时间。由于访存速度远远大于 CPU 的执行速度，尤其是批量访存指令，一旦遇到存储器分体冲突，将等待更长的时间。而在 ARM7TDMI、ARM9TDMI 这些没有 cache 的微处理器内核中，批量访存指令的 CPU 时间公式就完全变成如下形式：

$$\text{CPU time} = \text{IC} \times \text{CPIexecution} \times \text{Clock cycle time} + \text{Miss penalty} \times \text{Clock cycle time}$$

因此，在这些处理器内核中在处理诸如任务切换和进入中断的现场保护的批量访存指令时，系统将等待，从而影响实时性。

3.2 中断现场保护的优化策略

中断现场保护中，保护返回地址、程序状态字、堆栈指针是必需的，否则中断结束后将无法顺利返回。而保护通用寄存器的目的在于防止用户中断服务子程序使用其中的寄存器，造成对原有内容的覆盖而在中断返回后任务执行出错。因此在中断里对通用寄存器的保护完全可以取决于中断服务子程序对通用寄存器的使用情况，仅仅保存中断服务子程序中所用到的有限的几个通用寄存器，而不必保存所有通用寄存器。以 ARM 体系结构为例，在用户模式下可用的通用寄存器为 R0~R12，R13 用作堆栈指针、R14 为返回地址、R15 用作 PC，如果在中断服务子程序中只用到 R0~R12 中的一小部分，则在中断到来时可以仅仅只保存通用存器中的这一小部分，从而能够减少访存时间，最终达到缩短中断响应提高中断实时性的目的。

在实际情况中，这种策略是具有可行性的。首先，每个中断服务子程序中所需要的通用寄存器是可知的。在使用汇编语言编写用户中断服务子程序时，所需要的通用寄存器由程序员控制，使用 C 语言则由编译器决定具体使用到哪几个通用寄存器。其次，在现有的嵌入式操作系统中，往往要求中断服务子程序尽可能的短小，例如在 Linux 中，把中断服务子程序分成 Bottom Half 和 Top Half。因此，在大多数中断服务子程序中并没有用到所保护的全部通用寄存器，造成对其余通用寄存器的多余保护。

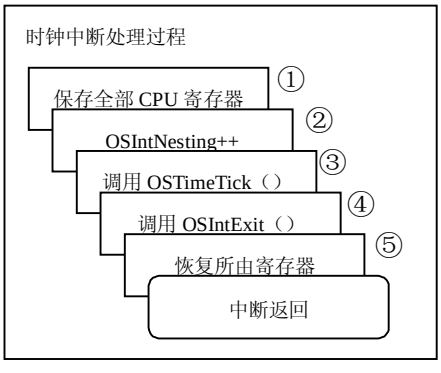

图 3　μC/OS-II 时钟中断处理

3.3 μC/OS-II 时钟中断现场保护优化

时钟中断是操作系统中比较重要的一个部分，也是实时性要求较高的部分，在 UNIX 中时钟中断的优先级定义为 6，仅次于最高优先级。以 μC/OS-II 时钟中断处理为例，中断处理过程如图 3。μC/OS-II 时钟中断服务中，首先要对中断嵌套计数器 OSIntNesting 进行加 1 操作，防止在嵌套的中断中进行任务调度；随后调用 OSTimeTick ()对每个睡眠任务的 OSTCBDly 进行减 1 以及对系统时间 OSTime 加 1 操作；最后调用 OSIntExit ()进行任务调度，如果不需要任务切换则返回到中断服务程序中。可见在时钟中断处理中，操作最多的集中在 OSTimeTick ()和 OSIntExit ()这两个函数上。通过 ARMCC 编译器的-s 选项对两者进行编译，在得到的汇编代码中，前者需要使用 R0、R1、R4-R7，后者需要 R0-R3，没有使用 R8-R12，而 OSIntNesting++的操作也完全可以使用 R0-R7 进行，这样，在进入中断处理时，需要保存的通用寄存器仅仅为 R0-R7。因此对图 3 中的①进行改写得到的保护中断现场的代码如图 4 所示。

```
STMFD    SP!,{LR}
STMFD    SP!,{R0-R7}
MRS      R0,CPSR_c
STMFD    SP!,{R0}
```

图 4　μC/OS-II 时钟中断现场保护

μC/OS-II 其他的中断处理与时钟中断相似，仅仅需要把 OSTimeTick ()替换成对应的处理，如果能在不牺牲代码效率的情况下，将相应处理集中到 R0-R3 这几个寄存器中，则该中断处理中，仅仅使用 R0-R3，只要对它们进行保护即可，从而能更进一步缩短中断响应时间，大大缩短中断响应时间，提高中断实时性。

4 总结

传统的中断现场保护保存所有寄存器的内容，虽然使得程序的设计得到简化，但是同时造成了多余的寄存器保护，增加了中断响应的时间。有限中断现场保护策略能够根据具体中断服务中所需要使用的通用寄存器而进行相应的有限的保护，缩短了现场保护的时间，使用户中断服务能够尽早的到处理，提高了中断实时性。但是同时，有限中断现场保护的效率也受到中断服务处理复杂程度以及编译器性能的影响。对于中断服务简单，而对实时性要求高的中断，效果较明显，而对于复杂的中断服务，需要的通用寄存器较多，从而中断现场保护的寄存器也较多。而在同一中断服务中，效率高的编译器能够在不牺牲代码效率的情况下使用尽可能少的寄存器来完成中断服务，从而减少了需要中断现场保护的寄存器数，达到提高中断实时性的要求。

参考文献

[1]Jean J. L，μC/OS-II, The Real-Time Kernel，R&D Technical Books, 1998

[2]John L.H，David A.P，计算机体系结构：量化研究方法：第 3 版，机械工业出版社，2002

[3]毛德操，胡席明，嵌入式系统：采用公开源代码和 StrongARM/Xscale 处理器，浙江大学出版社，2003

作者简介：钱贾敏 (1980-)，男 (汉族)，上海，硕士研究生，主要研究方向：嵌入式系统应用与研究；电话：021-65982980、13916637695;E-Mail: priest@sh163.net;王力生 (1954-)，男 (汉族)，上海，副教授，主要研究方向：嵌入式系统、电子商务方向.

(200092 上海同济大学计算机科学与技术系)钱贾敏　王力生

(Department of Computer Science and Technology, Tongji University, Shanghai 200092, China) Qian,Jiamin　Wang, Lisheng

联系方式：

(200030　上海市徐汇区天钥新村 54 号 501 室)钱贾敏

(收稿日期：2004.5.20)

技术创新