第35卷 第14期 计 算 机 工 程 2009年7月
Vol.35 No.14 Computer Engineering July 2009
·博士论文· 文章编号：1000—3428(2009)14—0020—02 文献标识码：A 中图分类号：TP316.2

嵌入式内核的任务上下文保护机制

段星辉，代作晓
(中国科学院上海技术物理研究所，上海 200083)

摘 要：单粒子翻转可能发生在内核堆栈时，破坏任务的上下文环境，从而导致星载软件运行结果出错、跑飞甚至崩溃。针对该情况，对嵌入式 μC/OS-II 内核进行改进，实现任务上下文保护机制。经实验验证，改进的内核能有效地克服单粒子翻转对内核堆栈造成的影响。
关键词：μC/OS-II 内核；任务上下文；单粒子翻转

Protection Mechanism for Task Context of Embedded Kernel

DUAN Xing-hui, DAI Zuo-xiao
(Shanghai Institute of Technical Physics, Chinese Academy of Sciences, Shanghai 200083)

【Abstract】 A single event upset may happen in the task stack and spoile the task context, so the on-board software may be affected in the way of getting wrong results, running to undefined point or becoming collapsed. Aiming at this problem, this paper improves the embedded kernel μC/OS-II and implements protection mechanism for task context. A test is given, which proves the protection mechanism for task context can effectively get rid of the effect on task stack caused by a single event.
【Key words】 μC/OS-II kernel; task context; single event upset

1 概述

随着星载计算机软件规模的逐渐增大，传统的针对单任务或少量任务的软件结构已很难满足目前星载软件的要求。嵌入式内核的应用简化了应用程序的设计与实现，增强了软件的扩展性和维护性，提高了系统的可靠性。常用的星载嵌入式操作系统有 VxWorks, pSOS, RTLinux 和 μC/OS-II 等。通过对几种操作系统的比较，结合自身项目需求，笔者在某星载仪器管理系统内部使用了 μC/OS-II 内核。

任务上下文(task context)是嵌入式操作系统中的一个重要概念，主要指在某个特定时间点以前的执行历史中尚未完成的过程，以及用于这些过程的局部变量在这个时间点上的映象[1]，基于 μC/OS-II 内核的任务堆栈的内容反应了这些信息。每个任务都有自己独立的堆栈，用于保存“中断”(硬件中断或因某种原因的任务挂起)时刻任务的上下文环境，以便恢复运行时能返回到被中断时刻的运行状态。如果保存任务上下文的堆栈内容遭受破坏，那么一旦该任务重新调度运行，它将无法返回中断前的运行状态，从而导致程序运行出错、程序跑飞甚至软件系统崩溃。在地面环境中，任务堆栈因外部因素而遭受破坏的可能性很小，但在太空环境，由于受空间高能粒子的冲击，星载计算机有可能发生单粒子翻转事件，任务堆栈的内容有可能因此遭到破坏。

2 内核任务切换分析

在 μC/OS-II 中创建的任务具有唯一的优先级，内核任务调度基于任务的优先级。μC/OS-II 是抢占式嵌入式操作系统，它总是运行进入就绪态任务中优先级最高的任务。确定哪个任务优先级最高就该哪个任务运行，这一工作是由调度器完成的。当调度器决定挂起当前运行任务转而运行其他任务时，就必须发生任务切换。任务切换表示：将被挂起任务的处理器寄存器推入自身的任务堆栈，然后将即将运行的任务的寄存器值从其栈中恢复到处理器寄存器。在 μC/OS-II 中，宏 OS_TASK_SW()用来执行任务级的任务切换。在内核移植时，针对使用的目标处理器，应该明确保存哪些变量和寄存器，从而确立任务堆栈结构。以 ARM7 处理器为例(笔者开发的某星载仪器管理系统使用 ARM7 作为控制器)，须保存的寄存器有 R15(PC), R14(LR), R12-R0, CPSR 和变量 OsEnterSum。OS_TASK_SW()保存当前任务 CPU 寄存器值的过程见图 1。

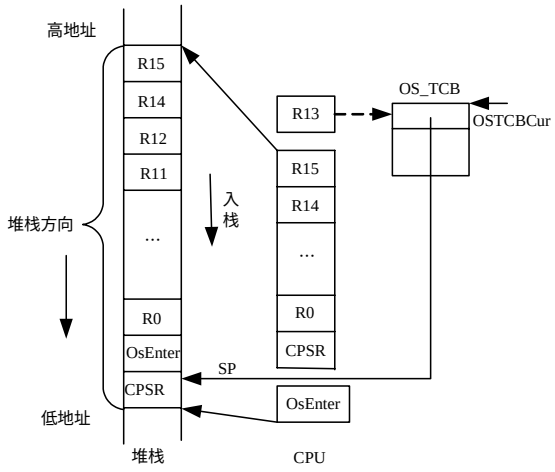

图 1 保存当前任务 CPU 寄存器值的过程

将当前 CPU 寄存器依次保存到当前任务的堆栈中，全部 17 个寄存器(包括 OsEnterSum)依次入栈后，此时 CPU 的 SP(R13)寄存器和 OSTCBCur->OSTCBStkPtr 都指向当前任务堆栈的同一个位置(都指向 CPSR)。

基金项目：国家“863”计划基金资助项目“超高光谱分辨率红外光谱仪”(2007AA12Z179)
作者简介：段星辉(1981－)，男，博士研究生，主研方向：嵌入式系统，仪器管理系统；代作晓，副研究员
收稿日期：2008-11-13 **E-mail：**starlight_ustc@yahoo.com.cn

当调度器确定下一个要运行的任务 OSTCBHighRdy 时，它被赋给 OSTCBCur，首先找到 OSTCBCur->OSTCBStkPtr，即重新开始运行的任务的堆栈指针，装入 CPU 的 R13 寄存器中，它指向堆栈中的寄存器 CPSR，然后按相反的方向从堆栈中的弹出之前保存的寄存器，整个过程如图 2 所示。最后通过将 R15(PC)弹出到 CPU 的 PC 寄存器中，重新开始运行的任务代码从 PC 指向的那一点开始运行，于是切换到新任务代码的过程完成[2-4]。

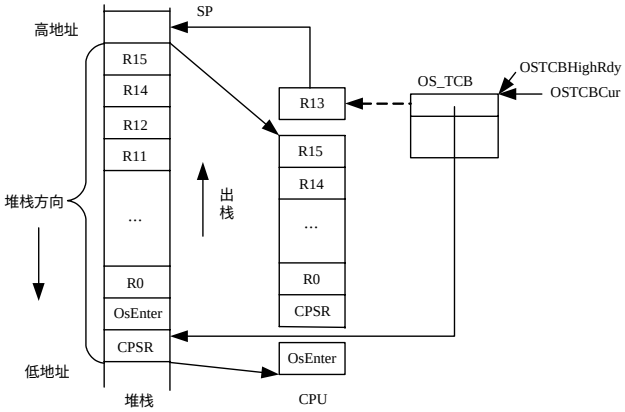

图 2　重新装入要运行任务的过程

3　任务上下文保护机制

在以上重新装入要运行的任务过程中，如果任务堆栈的内容在装入前已发生了变化，比如受高能粒子辐射发生单粒子翻转事件，那么重新装入 CPU 寄存器的值就不完全是上一次任务切换时保存下来的值了。任务的上下文环境遭受破坏，这样任务重新投入运行时，程序运行可能不正确、跑飞甚至造成系统崩溃，因此，对其进行保护是很有必要的。

为消除任务堆栈内容发生变化对程序运行造成的影响，笔者采用冗余思想，重新构建任务堆栈结构，对 μC/OS-II 的任务切换宏 OS_TASK_SW()进行了改进。具体方法为：保存任务下文环境(寄存器组和局部变量)时，将它们在堆栈中作 3 次拷贝，如图 3 所示；在恢复上下环境时，对堆栈中 3 个拷贝值进行表决：当 2 个值相同时，选择其一，忽略第 3 个值；如果三者不一致，则重启系统，如图 4 所示。这种机制为任务上下文保护机制。

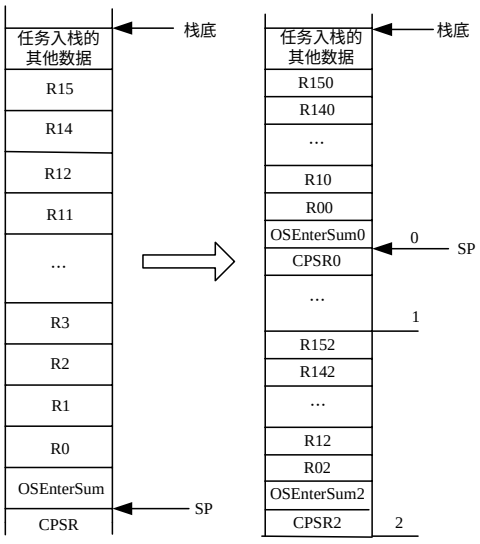

图 3　改进前后的任务堆栈结构

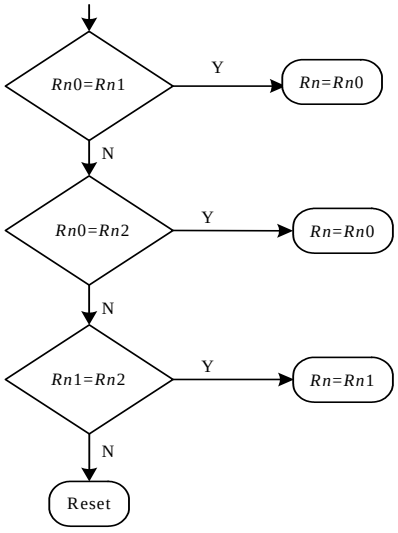

图 4　恢复上下文环境的流程

上下文环境假设为 *Rn*，其 3 个拷贝值 *Rn*0, *Rn*1, *Rn*2 存储在任务堆栈中。对内核需要修改的代码有 OS_CPU_C.C 文件中的初始化任务堆栈函数 OSTaskStkInit()和 OS_CPU_A.S 件中的上下文切换函数 OS_TASK_SW()。在 OSTaskStkInit()中，原内核代码只初始化 *Rn* 的值，现在应初始化 *Rn*0, *Rn*1 和 *Rn*2 的值，且应保持这 3 个寄存器或变量初始值相同。注意该函数最后返回的栈顶指针 SP 与未改进之前代码返回的值相同(指向 CPSR0，如图 3 所示的 SP)，而不是返回真正的栈顶指针(指向 CPSR2)。这样做的好处是保持原有内核代码框架，减少对原内核代码的修改；在 OS_TASK_SW()中，保存上下文环境时，增加拷贝代码，即把 *Rn*0 拷贝到 *Rn*1 和 *Rn*2 中。在恢复上下文时，*Rn*0, *Rn*1, *Rn*2 这 3 个值按图 4 所示的流程进行表决。

该堆栈保护机制是用时间和空间来换取内核的可靠性。同改进前内核代码相比，改进的内核代码要多保存 2 个进程上下文，且恢复上下时，要单个寄存器表决恢复，效率不高，从而一定程度上增加了任务切换时间。初始化堆栈代码和进程上下文切换代码相比改进前体积有了增加，尤其是后者，单个寄存器表决恢复代码使上下文切换代码变得臃肿，因此，内核代码存储空间需求加大。由于每个任务的堆栈应该多容纳 2 个上下文，对控制器 ARM7 来说，一个上下文为 17 个寄存器(包括 OSEnterSum 在内)，因此每个任务多需要 136 Byte，用户创建的任务越多，额外消耗的任务堆栈空间也越大。

时间和空间 2 个额外需求相对现在高速处理器和大容量内存来说，对整个系统的影响是微乎其微的。在航天航空领域，为了内核和系统的可靠性，这样的代价值得的。

4　实验与分析

为验证上述任务上下文保护机制(即堆栈保护机制)能有效消除单粒子翻转事件对任务堆栈的影响，本文创建一个用户任务，模拟单粒子翻转事件，对各任务的堆栈注入故障，即随意改变任务堆栈的内容。该任务代码如下：

```
void test(void *pdata)
{
  int i;
  pdata=pdata;
```

(下转第 24 页)