# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | CPU 异常处理与上下文切换 |
| 姓 名： | 江姝潼 |
| 学 号： | 2019211653 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 刘健培、戴志涛 |

2021年 11 月 24日

# 实验目的

* 了解开发环境的使用。
* 学会通过查阅文档和数据手册获取信息。
* 掌握 cortex-M4 体系结构中核内寄存器和异常处理的使用方式。
* 理解处理器异常上下文切换的实现方式。
* 了解系统调用的实现方式。
* 掌握基本的软件编写与调试方式。

# 实验环境

* FS-STM32F407开发平台
* ST-Link 仿真器
* RealView MDK5.23集成开发软件
* 串口调试工具
* PC机Window7/8/10 (64bit)

# 实验要求

1. 本实验通过编写系统调用理解 Cortex-M 处理器异常处理的特点与流程。

2. 在给定的参考代码基础上，参考并分析已实现的系统调用框架，实现一个完整的系统调用函数。

3.指定：基于 SVC 指令实现一个提升特权的系统调用。即首先将程序设置到用户模式，然后调用此系统调用，通过异常进入特权模式，在特权模式中将程序权限提升到特权模块，异常返回后，用户程序将以特权模式运行。

自拟：系统调用的功能自拟，例如，可以读取系统 Systick 时间、调用内核 API、控制任务状态、操作外设、控制用户 login 等等，简单的也可以打印输出、做简单的数学运算等等。

4. 编写一个测试程序验证所编系统调用成功执行。

# 实验原理

描述实验原理。

如：硬件接口的控制原理、所使用的算法的原理等。

1. SYCall异常

使用 svc 系统调用指令，触发 SVCall 异常。进入异常时，硬件会切换模式，并保存必要的寄存器。然后异常处理程序再根据需要保存需要的额外寄存器。退出异常时，流程基本与进入异常时相反，首先软件需要恢复部分寄存器，然后通过机器指令让硬件恢复之前硬件保存的寄存器，并跳转到异常发生时的地址继续执行。

1. 系统调用

系统调用（system call），指运行在用户空间的程序向操作系统内核请求需要更高权限运行的服务。系统调用提供用户程序与操作系统之间的接口。系统调用和普通库函数调用非常相似，只是系统调用由操作系统内核提供，运行于内核核心态，而普通的库函数调用由函数库或用户自己提供，运行于用户态。

Linux 在 x86 上的系统调用通过 int 80h 实现，用系统调用号来区分入口函数。操作系统实现系统调用的基本过程是：

1) 应用程序调用库函数（API）；

2) API 将系统调用号存入 EAX，然后通过中断调用使系统进入内核态；

3) 内核中的中断处理函数根据系统调用号，调用对应的内核函数（系统调用）；

4) 系统调用完成相应功能，将返回值存入 EAX，返回到中断处理函数；

5) 中断处理函数返回到 API 中；

6) API 将 EAX 返回给应用程序。

应用程序调用系统调用的过程是：

1) 把系统调用的编号存入 EAX；

2) 把函数参数存入其它通用寄存器；

3) 触发 0x80 号中断（int 0x80）。

在嵌入式系统环境，嵌入式操作系统与用户应用程序往往链接在一个可执行文件中，往往也不区分用户程序与操作系统内核的权限。有些嵌入式操作系统也提供系统调用作为一个可配置的选项，大部分则默认将用户应用程序作为特权任务运行。

1. Cortex-M 切换处理器特权级别

当程序运行在用户态时，权限是受限的。在线程模式＋用户级下，对系统控制空间（SCS）的访问将被阻止。除此之外，还禁止使用 MSR 访问特殊功能寄存器。

在特权级下的代码可以通过置位 CONTROL[0]来进入用户级。而不管是任何原因产生了任何异常，处理器都将以特权级来运行其服务例程，异常返回后将回到产生异常之前的特权级。用户级下的代码不能再试图修改 CONTROL[0]来回到特权级。它必须通过一个异常 handler，由那个异常 handler 来修改CONTROL[0]，才能在返回到线程模式后拿到特权级。

CONTROL［0］只有在特权级下才能访问。用户级的程序如想进入特权级，通常都是使用一条“系统服务呼叫指令（SVC）”来触发“SVC 异常”，该异常的服务例程可以选择修改 CONTROL[0]。

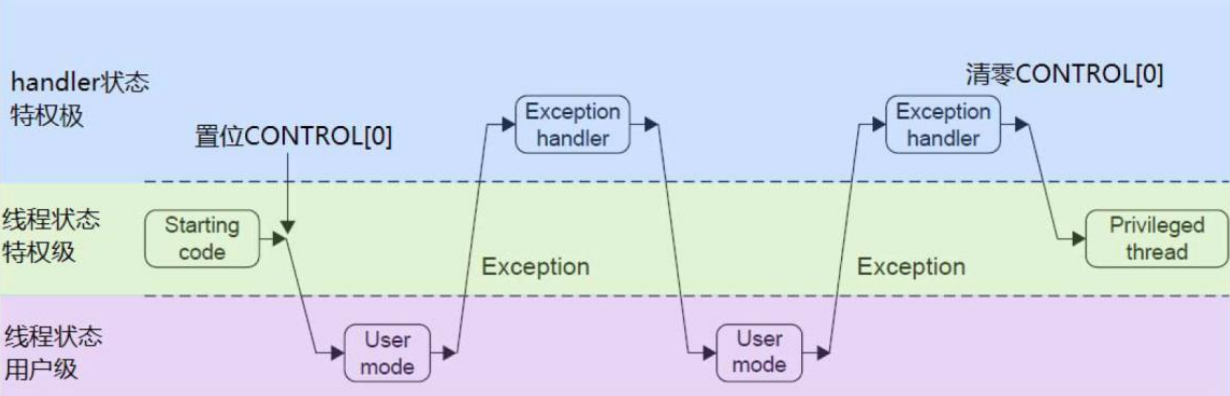


图1- 1 特权级和处理模式的改变

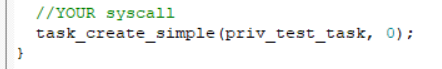
# 实验步骤

1. 编写系统调用函数

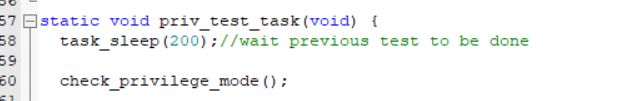
申明用户侧系统调用函数和系统调用号。在syscall.h 中申明用户侧函数 raise\_privilege，并定义其使用的系统调用号SYS\_raise\_privilege，并在 syscall\_ccarm.S 中使用 EQU 伪指令定义系统调用号（与 syscall.h 中定义的值相同），并使用宏 SYSCALL 实现 raise\_privilege 汇编函数体，最后在在 syscall.c 中的sys\_call\_table[]表中增加表项。上述过程在老师所给的代码中已经完成。

1. 编写系统调用程序

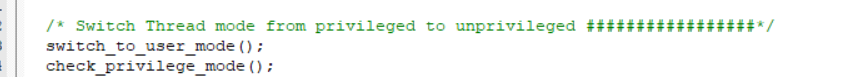
在project1\_main函数中找到自己编写的系统调用的入口，在task\_create\_simple中，执行的内容是priv\_test\_task，在该函数中实现用户态和特权态的相互切换。

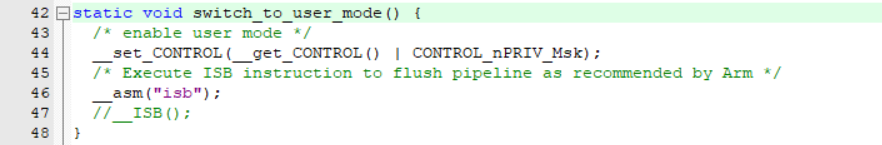


首先创建一个运行在特权等级的任务，用sleep等待任务执行完毕，用check\_privilege\_mode()查看目前所处的状态，在后面的结果中可以看到处于特权态。

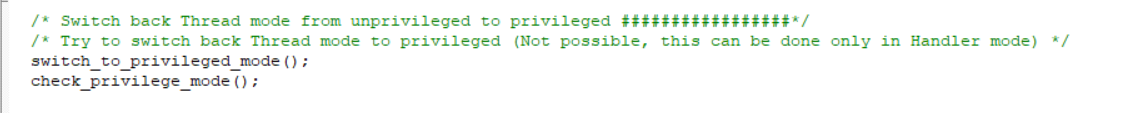


接下来用switch\_to\_user\_mode()函数切换至用户态，查看操作，可以看到是通过设置 CONTROL［0］=1切换的，原因是任务在特权态是可以访问 CONTROL［0］的。



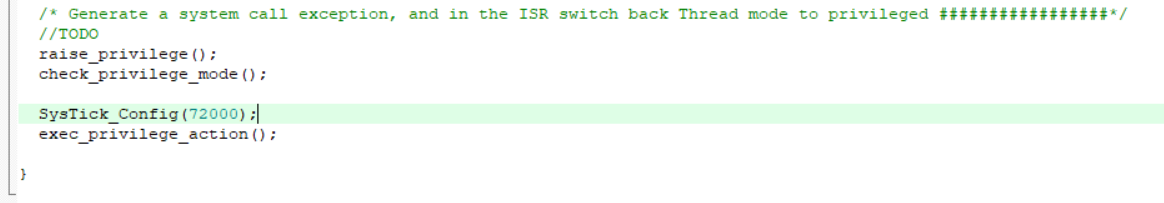


下面调用switch\_to\_privileged\_mode()将自身切换到用户态，并用check\_privilege\_mode()查看目前所处的状态，根据后面结果可知，还是处于特权态，原因是此时任务处于用户态，无法调用特权操作。如果直接调用切换特权的函数，是没有任何效果的。



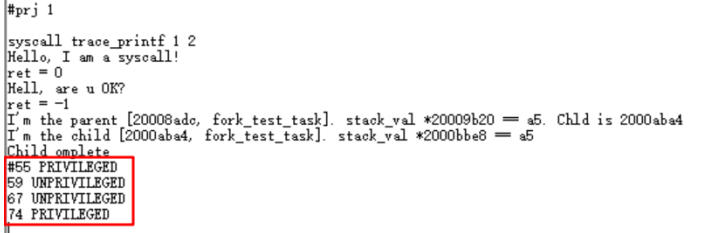
如果按照原代码执行exec\_privilege\_action()，即调用特权指令，会导致 HardFault 异常，程序运行体现在没有任何输出，处于卡死的状态。

我们的操作是调用raise\_privilege()提升状态，将用户态切换为特权态，然后执行在特权态才能执行的操作，包括指定的exec\_privilege\_action()函数和系统调用的功能自拟，这里是读取系统 Systick 时间，该操作只能在特权状态下执行。



3. 测试输出结果

在 sscom 中输入 prj 1命令，将调用 project1\_main 函数，运行结果如下，符合上述预期。



# 实验方案与实现

## 软件结构

如：**框图**说明软件系统结构，文字说明系统各模块重要的细节定义（常量变量定义、数据结构、主要函数等）。

在MDK环境中，点击启动调试并打开串口监视器查看串口输出。首先执行reset\_handler来启动开发板，并调用systemInit进入main初始化各外设，接下来初始化并启动多任务调度，在使用控制台程序中等待成用户输入命令。在本实验中我们输入命令prj 1,来执行project1\_main函数， 在priv\_test\_task任务中调用刚刚书写的各个操作，包括查看当前的状态，并在特权态下转换成用户态，再从用户态转化为特权态，并调用特权态下才能执行的指令操作，观察当前是否能否正常进行。



图1- 2 执行流程

## 源代码

static void priv\_test\_task(void) {

    task\_sleep(200);//wait previous test to be done

    check\_privilege\_mode();

    /\* Switch Thread mode from privileged to unprivileged #################\*/

    switch\_to\_user\_mode();

    check\_privilege\_mode();

    /\* Unprivileged access mainly affect ability to:

     - Use or not use certain instructions such as MSR fields

     - Access System Control Space (SCS) registers such as NVIC and SysTick \*/

    /\* Switch back Thread mode from unprivileged to privileged #################\*/

    /\* Try to switch back Thread mode to privileged (Not possible, this can be done only in Handler mode) \*/

    switch\_to\_privileged\_mode();

    check\_privilege\_mode();

    //uncomment this will cause HardFault

    //exec\_privilege\_action();

    /\* Generate a system call exception, and in the ISR switch back Thread mode to privileged #################\*/

    //TODO

    raise\_privilege();

    check\_privilege\_mode();

    SysTick\_Config(72000);

    exec\_privilege\_action();

}

# 实验结果与分析

1.请描述系统调用的过程（从用户模式下开始系统调用到系统调用完成）。在此过程中，处理器的状态与进入异常处理时的堆栈是如何变化的？

调用系统调用，实质上是去调用一个异常指令 svc，触发处理器的异常处理过程，切换处理器到内核态，在此期间将原函数对应的系统调用号和参数通过堆栈传入到内核态，内核态中查表调用真正的实现函数，并将结果通过异常返回。所以用户态程序无法控制调用的子函数如何具体实现，只能通过传递信息的方式请求系统去完成该函数的功能。系统调用相当于一扇门，隔开了用户态/非特权级和内核态/特权级，使得敏感的特权指令在内核程序的控制之下，用户态的应用程序无法直接执行特权指令（否则 fault 异常伺候），需要通过这个“门”“喊话（call）”，引起内核态的“注意”，才能委托内核完成功能。这是处理器支持操作系统特权的一种系统机制。

当异常产生且被处理器接受时，压栈流程会将寄存器压入栈中并组织栈帧，如图1- 3所示。

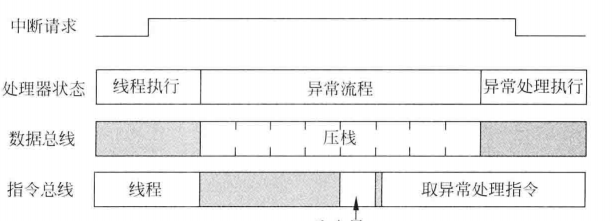


图1- 3 压栈和取向量

Cortex-M3和Cortex-M4处理器具有多个总线接口。在压栈操作的同时，处理器还可以开始取向量和取指。这样，由于压栈操作可以和Flash存储器访问同时进行，哈佛总线架构可以降低中断等待时间。若向量表位于SRAM或异常也位于SRAM，中断等待会稍微增加。

压栈操作中用的栈可以为主栈（使用主栈指针，MSP）或进程栈（使用进程栈，PSP）。若处理器运行在线程模式且使用MSP（CONTROL寄存器的第0位为0，默认配置），则压栈操作在执行时使用主栈MSP；处理器运行在线程模式且使用进程栈（CONTROL寄存器的第1位为1），则压栈操作执行时使用进程栈PSP。在进入处理模式后，处理器必须使用MSP，所有嵌套中断的压栈操作执行时都使用主栈MSP。

2.系统调用的返回值是何返回给调用者的？

在异常处理结束时，异常入口处生成的EXC\_RETURN数值的第2位用于确定提取栈帧时所用的栈指针。若第2位为0，则处理器会知道之前压栈时使用的是主栈；若第2位为1，处理器就会知道压栈时使用的是进程栈。

确定了系统调用时所使用的战史针，就可以回到对应调用前所在的栈，并通过读取栈中保存的各寄存器的值获取执行后的返回值。

1. 实验中实现系统调用的方式最多能传递几个参数？为什么？

用于ARM架构的C编译器遵循ARM的一个名为AAPCS的规范。根据这份标准，C函数可以修改RO~R3、R12、R14（LR）以及PSR。若C函数需要使用R4~R11，就应该将这些寄存器保存到栈空间中，并且在函数结束前将它们恢复。

RO~R3、R12、LR以及PSR被称作“调用者保存寄存器”，若在函数调用后还需要使用这些寄存器的数值，在进行调用前，调用子程序的程序代码需要将这些寄存器的内容保存到内存中（如栈）。函数调用后不需要使用的寄存器数值则不用保存。

R4~R11为“被调用者保存寄存器”，被调用的子程序或函数需要确保这些寄存器在函数结束时不会发生变化（与进入函数时的数值一样）。这些寄存器的数值可能会在函数执行过程中变化，不过需要在函数退出前将它们恢复为初始值。

一般来说，函数调用将R0~R3作为输入参数，R0则用作返回结果。

4.实验中使用的教学 RTOS 进行任务切换时，需要切换处理器状态（即核内寄存器），切换函数也是使用异常（pendsv 异常）实现的，是否理解通过 pendsv异常进行处理器状态切换的原理与过程？

PendSV是为系统设备而设的“可悬挂请求”（pendable request）。SVC异常是必须立即得到响应的（若因优先级不比当前正处理的高，或是其它原因使之无法立即响应，将上访成Hardware fault），应用程序执行SVC 时都是希望所需的请求立即得到响应。上下文切换 不能在中断中进行，会导致中断延期。为了解决这个问题，使用 PendSV。PendSV可以挂起，也就是等到别的 ISR结束后缓期执行。为了实现缓期执行PendSV，PendSV一定要被设置为最低优先级的异常，系统可以利用它“缓期执行”一个异常——直到其它重要的任务完成后才执行动作。

挂起PendSV 的方法是，软件实现OSIntCtxSw()函数，向NVIC 的PendSV 悬起寄存器中写1。

# 实验总结

第一次做嵌入式实验，开始有点迷茫，后来认真看了实验指导书后有了些眉目。实验的关键在于根据实验要求，学会查找指导手册获取自己需要的信息，并据此获得对应的信息用于判断处理器分别处于何种模式（handler or thread）、使用何种栈（MSP or PSP）和何种特权等级（特权与非特权）。

在现场做实验的时候，想要实现预期的结果并不困难，只需要写寥寥的几行代码来提升它的优先级就可以了。但在真正写实验报告、并进行后期复习的时候，才发现系统调用和异常处理的这个过程执行起来是非常精妙的，包括栈的保存和调用，以及各种模式的切换，这在现场做实验的时候是没有体会到的，只有在后期开始写实验报告的时候，才逐渐明白这些背后的原理。整体而言，这个实验带给我的收获是非常丰富的，让我了解了在扩大M3中这异常和中断这一重要操作的执行过程以及具体细节，并锻炼了我查阅文献来获取信息的能力。