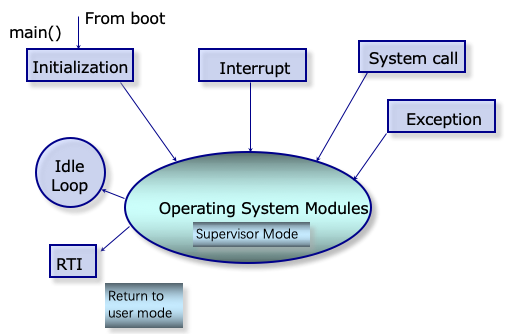
**实验四 异常处理**

计科2302 王任杰 202308010212

中断、异常和陷阱指令是操作系统的基石，现代操作系统就是由中断驱动的。本实验和实验五的目的在于深刻理解中断的原理和机制，掌握CPU访问中断控制器的方法，掌握Arm体系结构的中断机制和规范，实现时钟中断服务和部分异常处理等。

**陷入操作系统**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id2)

如下图所示，操作系统是一个多入口的程序，执行陷阱（Trap）指令，出现异常、发生中断时都会陷入到操作系统。



**ARMv8的中断与异常处理**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#armv8)

**注意**

访问Arm官网下载并阅读 [ARM Cortex-A Series Programmer’s Guide for ARMv8-A](https://developer.arm.com/documentation/den0024/a/AArch64-Exception-Handling/Exception-handling-registers) 和 [AArch64 Exception and Interrupt Handling](https://developer.arm.com/documentation/100933/0100/AArch64-exception-vector-table) 等技术参考手册。

ARMv8 架构定义了两种执行状态(Execution States)，AArch64 和 AArch32。分别对应使用64位宽通用寄存器或32位宽通用寄存器的执行 [1](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id19) 。

上图所示为AArch64中的异常级别(Exception levels)的组织。可见AArch64中共有4个异常级别，分别为EL0，EL1，EL2和EL3。在AArch64中，Interrupt是Exception的子类型，称为异常。 AArch64 中有四种类型的异常 [2](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id20) ：

* Sync（Synchronous exceptions，同步异常），在执行时触发的异常，例如在尝试访问不存在的内存地址时。
* IRQ （Interrupt requests，中断请求），由外部设备产生的中断
* FIQ （Fast Interrupt Requests，快速中断请求），类似于IRQ，但具有更高的优先级，因此 FIQ 中断服务程序不能被其他 IRQ 或 FIQ 中断。
* SError （System Error，系统错误），用于外部数据中止的异步中断。

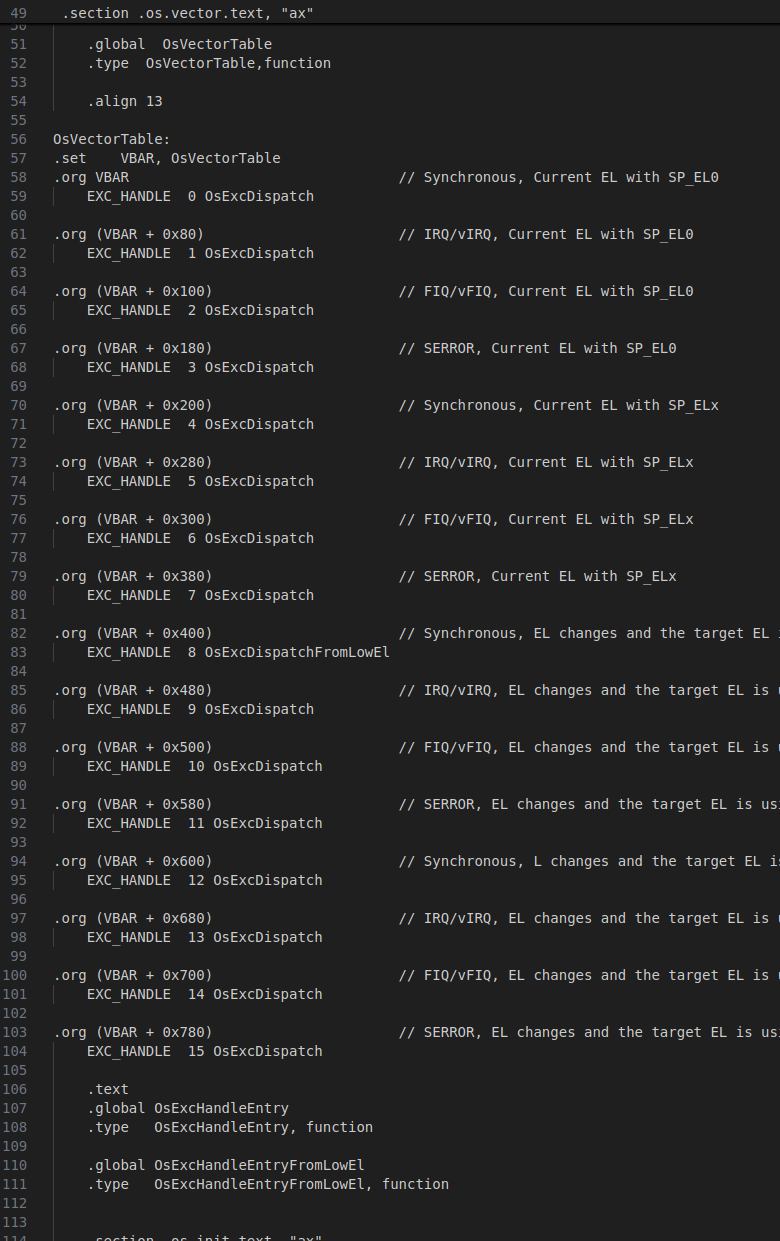
当异常发生时，处理器将执行与该异常对应的异常处理代码。在ARM架构中，这些异常处理代码将会被保存在内存的异常向量表中。每一个异常级别（EL0，EL1，EL2和EL3）都有其对应的异常向量表。需要注意的是，与x86等架构不同，该表包含的是要执行的指令，而不是函数地址 [3](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id21) 。

异常向量表的基地址由VBAR\_ELn给出，然后每个表项都有一个从该基地址定义的偏移量。 每个表有16个表项，每个表项的大小为128（0x80）字节（32 条指令）。 该表实际上由4组，每组4个表项组成。 分别是：

* 发生于当前异常级别的异常且SPSel寄存器选择SP0 [4](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id22) ， Sync、IRQ、FIQ、SError对应的4个异常处理。
* 发生于当前异常级别的异常且SPSel寄存器选择SPx [4](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id22) ， Sync、IRQ、FIQ、SError对应的4个异常处理。
* 发生于较低异常级别的异常且执行状态为AArch64， Sync、IRQ、FIQ、SError对应的4个异常处理。
* 发生于较低异常级别的异常且执行状态为AArch32， Sync、IRQ、FIQ、SError对应的4个异常处理。

**异常向量表**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id8)

新建 src/bsp/prt\_vector.S 文件，参照这里 [3](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id21) 定义异常向量表如下：

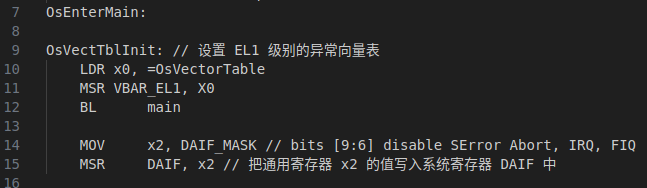


可以看到：针对4组，每组4类异常共16类异常均定义有其对应的入口，且其入口均定义为 EXC\_HANDLE vecId handler 的形式。

**提示**

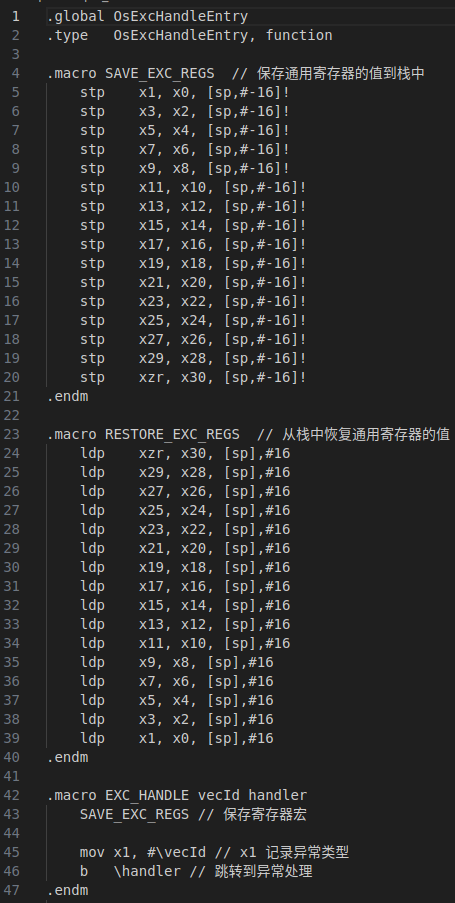
CPSR 寄存器中有当前栈的选择 bits[0] 0:SP\_EL0,1:SP\_ELX

在 prt\_reset\_vector.S 中的 OsEnterMain: 标号后加入代码



**上下文保存与恢复**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id10)

EXC\_HANDLE 实际上是一个宏，其定义如下。

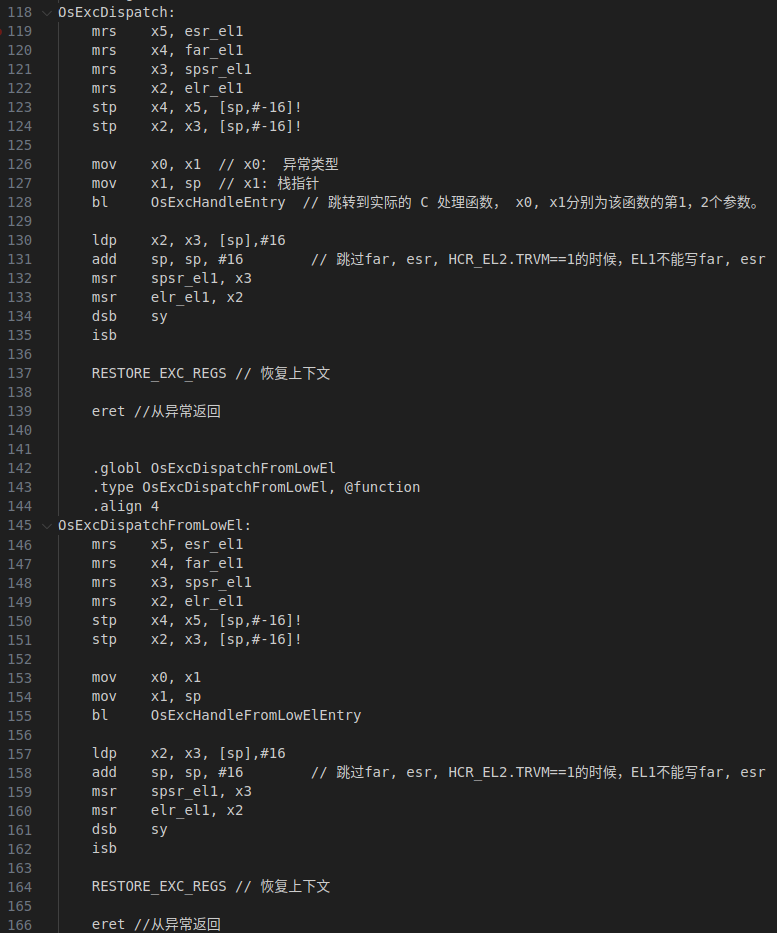


**提示**

注意把这部分代码放到 src/bsp/prt\_vector.S 文件的开头

EXC\_HANDLE 宏的主要作用是一发生异常就立即保存CPU寄存器的值，然后跳转到异常处理函数进行异常处理。

随后，我们继续在 src/bsp/prt\_vector.S 文件中实现异常处理函数，包括 OsExcDispatch 和 OsExcDispatchFromLowEl。

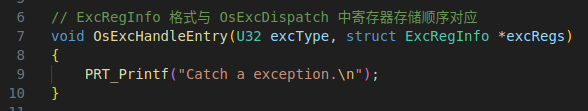


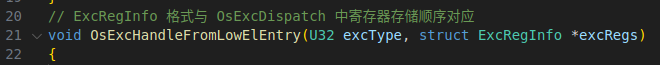
OsExcDispatch 首先保存了4个系统寄存器到栈中，然后调用实际的异常处理 OsExcHandleEntry 函数。当执行完 OsExcHandleEntry 函数后，我们需要依序恢复寄存器的值。这就是操作系统课程中重点讲述的上下文的保存和恢复过程。

OsExcDispatchFromLowEl 与 OsExcDispatch 的操作除调用的实际异常处理函数不同外其它完全一致。

**异常处理函数**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id11)

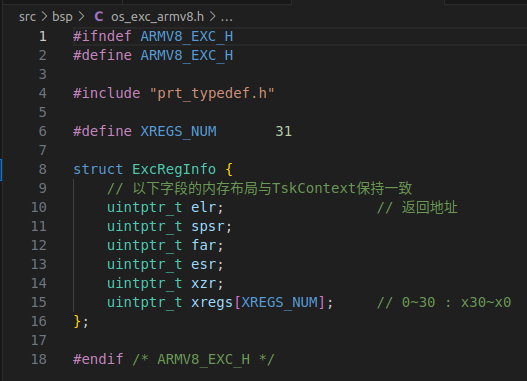
新建 src/bsp/prt\_exc.c 文件，实现实际的 OsExcHandleEntry 和 OsExcHandleFromLowElEntry 异常处理函数。





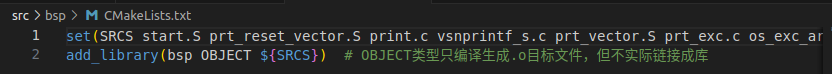
注意到上面两个异常处理函数的第2个参数是 struct ExcRegInfo \* 类型，而在 src/bsp/prt\_vector.S 中我们为该参数传递是栈指针 sp。所以该结构需与异常处理寄存器保存的顺序保持一致。

新建 src/bsp/os\_exc\_armv8.h 文件，定义 ExcRegInfo 结构。



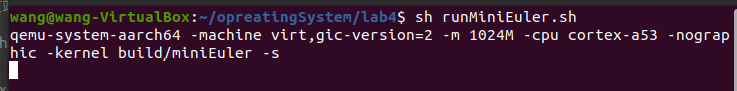
**提示**

注意把上面的新增文件加入构建系统。



**触发异常**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id12)

注释掉 FPU 启用代码，构建系统并执行发现没有任何信息输出，通过调试将会观察到异常。



**系统调用**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id13)

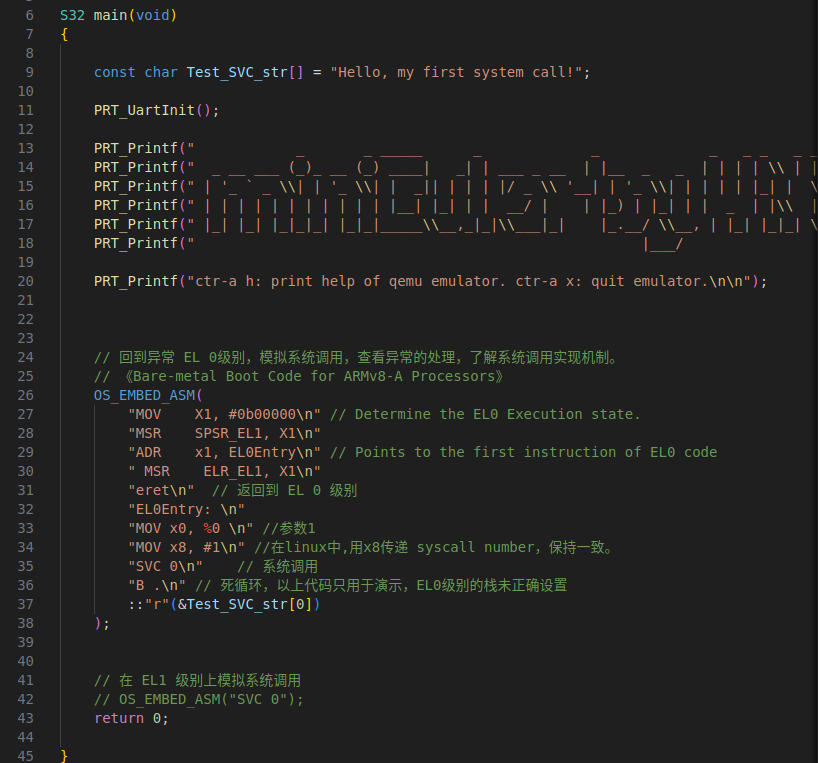
**提示**

下面请启用 FPU。

系统调用是通用操作系统为应用程序提供服务的方式，理解系统调用对理解通用操作系统的实现非常重要。下面我们来实现1条简单的系统调用。

EL 0 是用户程序所在的级别，而在lab1中我们已经知道CPU启动后进入的是EL1或以上级别。

在 main 函数中我们首先返回到 EL0 级别，然后通过 SVC 调用一条系统调用.



**备注**

OS\_EMBED\_ASM 在 prt\_typedef.h 中定义为 *\_\_asm\_\_ \_\_volatile\_\_*，用于 C 与 ASM 混合编程。

SVC 是 arm 中的系统调用指令，相当于 x86 中的 int 指令。

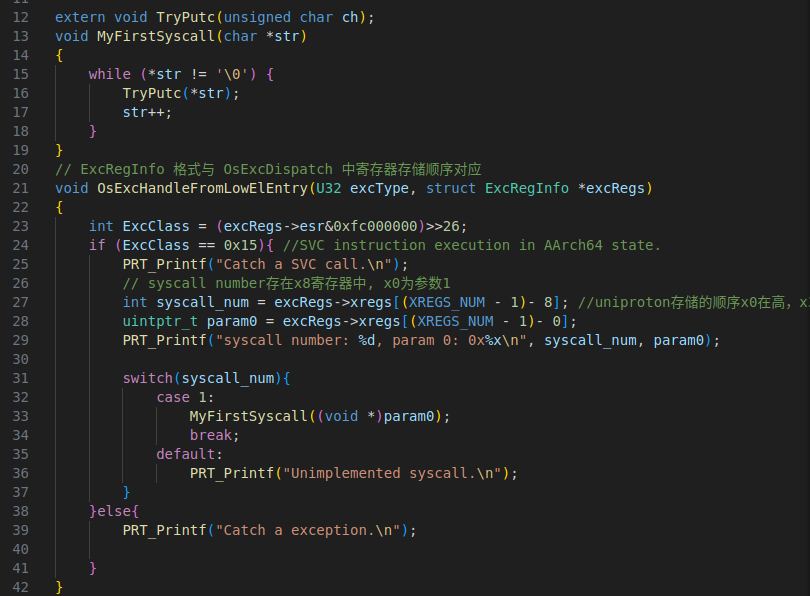
**备注**

汇编语法可以参考 GNU ARM Assembler Quick Reference [5](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id23) 和 Arm Architecture Reference Manual Armv8 (Chapter C3 A64 Instruction Set Overview) [6](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id24)

内联汇编中Clobbers的用途到底是什么？ [7](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id25)

**系统调用实现**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id17)

在 src/bsp/prt\_exc.c 修改 OsExcHandleFromLowElEntry 函数实现 1 条系统调用。

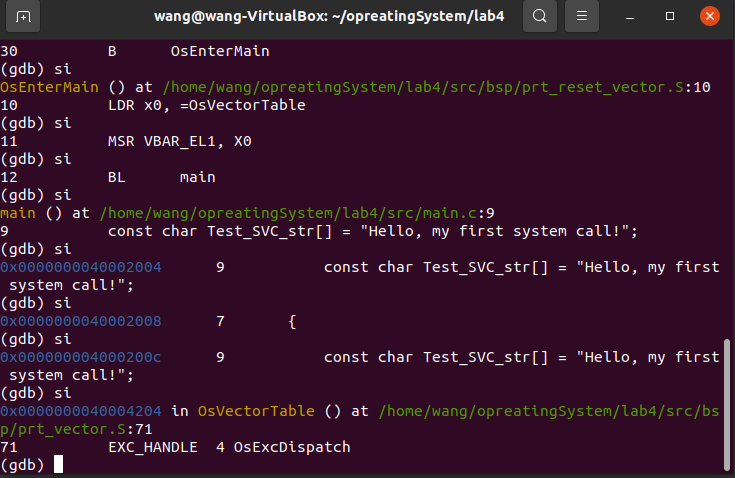


**lab4 作业**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#lab4)

**作业1**[**ℑ**](https://os2024lab.readthedocs.io/zh-cn/latest/lab4/index.html#id18)

查找 启用FPU 前异常出现的位置和原因。禁用FPU后PRT\_Printf工作不正常，需通过调试跟踪查看异常发生的位置和原因 elr\_el1 esr\_el1 寄存器









0001 1111 1110 0000 0000 0000 0000





