# PA4实验报告

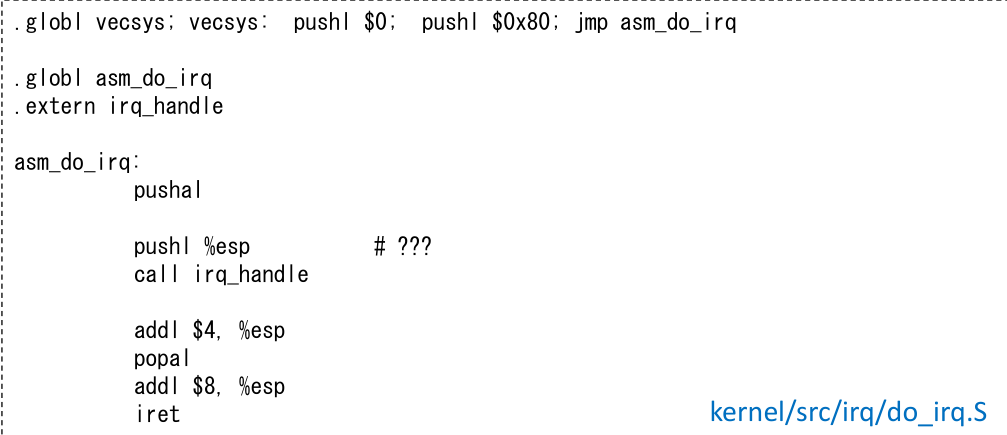
161220096 欧阳鸿荣

**1. 详细描述从测试用例中的 int $0x80 开始一直到 HIT\_GOOD\_TRAP 为止的详细的系统行为（完整描述控制的转移过程，即相关函数的调用和关键参数传递过程），可以通过文字或画图的方式来完成；**

执行int $0x80时，调用了int指令，通过解析操作码，获取中断号0x80，随后将其作为参数，调用raise\_sw\_intr()函数，该函数更新eip地址后，便调用raise\_intr()函数。

在raise\_intr函数中的intr\_no依然是0x80。随后，依次将eflags，CS和eip的值压栈，并从IDTR总读出IDT的首地址，根据中断号0x80在IDT中索引得到一个门描述符，把门描述符的段选择符装载入CS寄存器，接着调用load\_sreg函数加载cs的隐藏部分。根据段选择符中type的信息判断是中断还是陷阱。如果是中断便把IF清零。最后把offset赋给eip，raise\_intr的使命也就终结了

随后返回int指令，由于return 0，此时的eip便是中断处理程序的入口地址。执行到这一步后，便是操作系统（kernel）的工作了



通过入口地址的信息，跳转到kernel/src/irq/do\_irq.S的入口函数vecsys()，执行pushl $0 和 pushl $0x80后，压入错误码和异常号，跳转到asm\_do\_irq中，执行三个阶段：

（1）准备阶段：执行pushal和pushl %esp，在内核栈中保存各寄存器内容（现场信息）。代码将会把用户进程的通用寄存器保存到堆栈上, 这些寄存器的内容连同之前保存的错误码,以及eflags, CS, eip形成了trap frame

（2）处理阶段：执行call irq\_handle，调用函数irq\_handle，此时传入的是TrapFrame的变量\*tf，根据tf读出irq确定异常事件的类型，由于是0x80，kernel调用 do\_syscall()函数。

在do\_syscall()函数中，根据hello-inline中传入的参数tf->eax，为4，因此调用sys\_write函数，该函数根据tf指针把TrapFrame的ebx，ecx，edx传入fs\_write()函数调用，从而在屏幕输出hello world。

（3）恢复阶段：返回到do\_irq.S中，执行popa和iret，是恢复用户进程的现场, kernel将根据之前保存的trap frame中的内容, 恢复用户进程的通用寄存器，最后通过iret指令恢复用户进程的eip, CS, eflags。系统调用结束

此后，cpu.eip回到int指令的后一条指令，便继续执行hello-inline的代码

最后，便是喜闻乐见的HIT GOOD TRAP了

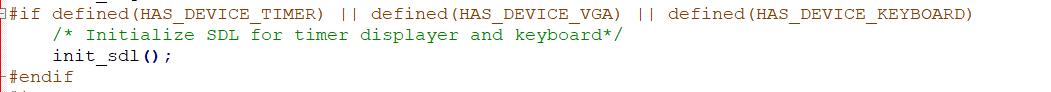
**2. 在描述过程中，回答 kernel/src/irq/do\_irq.S 中的 push %esp 起什么作用，画出在 call irq\_handle 之前，系统栈的内容和 esp 的位置，指出 TrapFrame 对应系统栈的哪一段内容。**

（1）push %esp的作用是把执行完pusha后的esp压栈，而这个esp指向的是TrapFrame的首地址，因此这个步骤是在把TrapFrame的指针作为参数传给irq\_handle

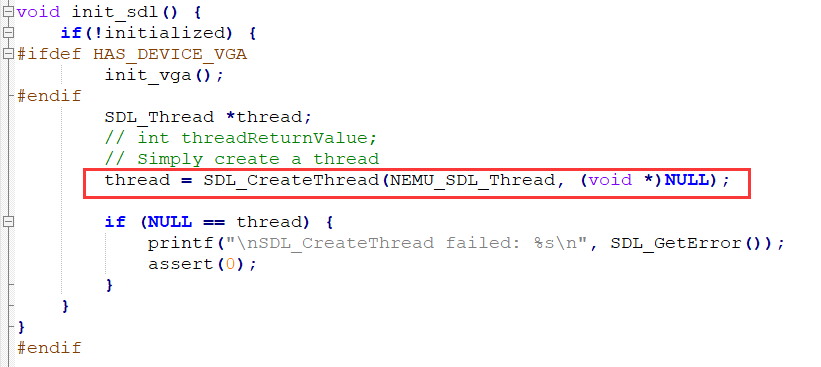
（2）在call irq\_handle之前，系统栈的内容如下：

|  |  |
| --- | --- |
|  | 返回地址 |
| ESP → | 保存的ebp |
|  | …… |
|  | eflags |
|  | CS |
|  | eip |
|  | 0 |
|  | 0x80 |
| TrapFrame | eax |
|  | ecx |
|  | edx |
|  | ebx |
|  | ebp |
|  | esi |
|  | edi |
| ESP → |  |

**3. 详细描述 NEMU 和 Kernel 响应时钟中断的过程和先前的系统调用过程不同之处在哪里？相同的地方又在哪里？可以通过文字或画图的方式来完成。**

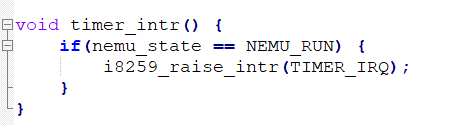
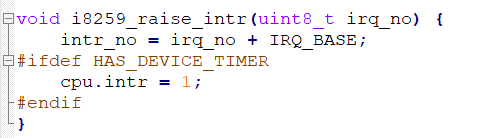


开启HAD\_DEVICE\_TIMER后，调用init\_sdl()函数，创建一个SDL线程

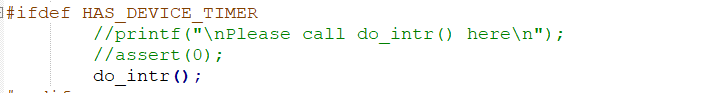


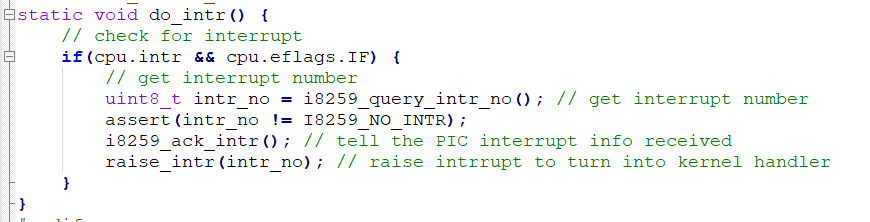
随后，在NEMU\_SDL\_Thread()中，调用了timer\_intr()函数，在这个函数中检测nemu的状态，若nemu尚处于运行状态，则不断通过i8259\_raise\_intr()函数提交时钟中断请求



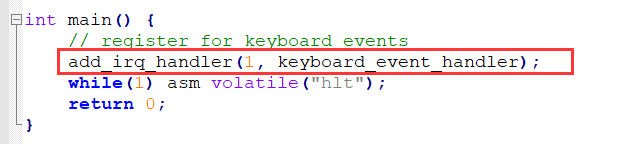


在cpu.c中，通过do\_intr()函数检测是否产生外部中断信号，如果产生时钟中断信号，则同样调用raise\_intr()函数，之后处理中断的过程同上的int $0x80中断过程，不过中断异常处理程序不同。

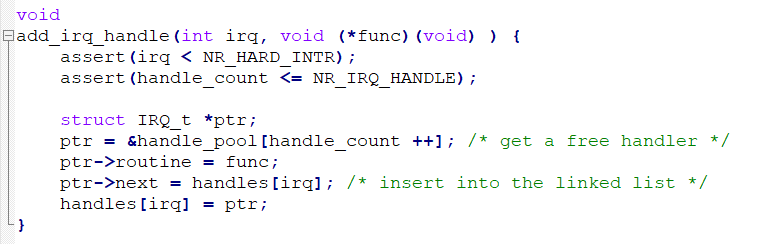




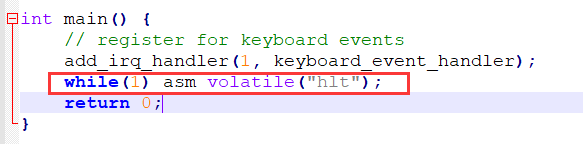
**4. 注册监听键盘事件是怎么完成的？**

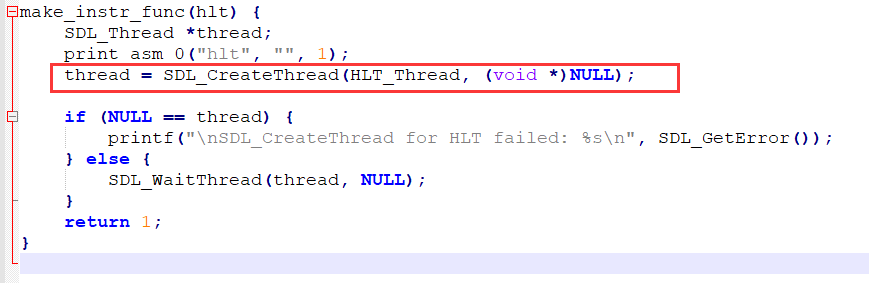


开启HAS\_DEVICE\_KEYBOARD后，在testcase/srt/echo.c中，main函数通过调用add\_irq\_handler，将IRQ\_t类型的指针存入handles数组，从而完成注册监听键盘事件。

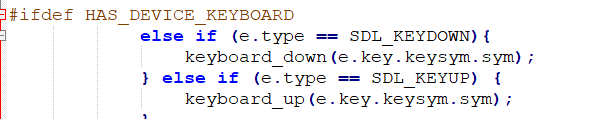


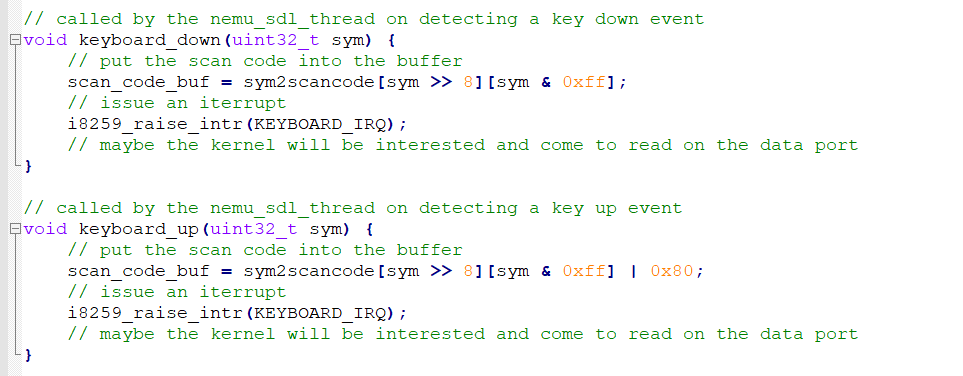
**5. 从键盘按下一个键到控制台输出对应的字符，系统的执行过程是什么？如果涉及与之前报告重复的内容，简单引用之前的内容即可。**





开启HAS\_DEVICE\_KEYBOARD后，完成注册监听键盘事件后，通过调用内联汇编hlt，创建并调用HLT\_Thread线程并检测SDL\_KEYDOWN和SDL\_KEYUP事件，一旦发生则调用相应的键盘中断处理函数





可以发现，和时钟中断类似，通过调用i8259\_raise\_intr函数，得到中断号，在cpu.c中农工的do\_intr函数中进入系统中断，类似的过程后，进入键盘处理程序，获取键盘值，然后调用printc()函数，通过int指令内部中断的方式调用系统函数sys\_write，过程同hello-inline，从而实现在控制台输出字符

