# PA4 实验报告

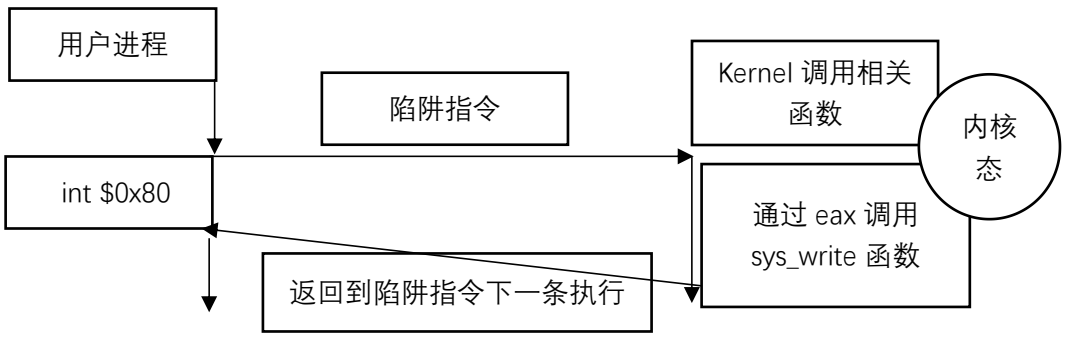
161220047

## PA4.1

### §4-1.3.1 通过自陷实现系统调用

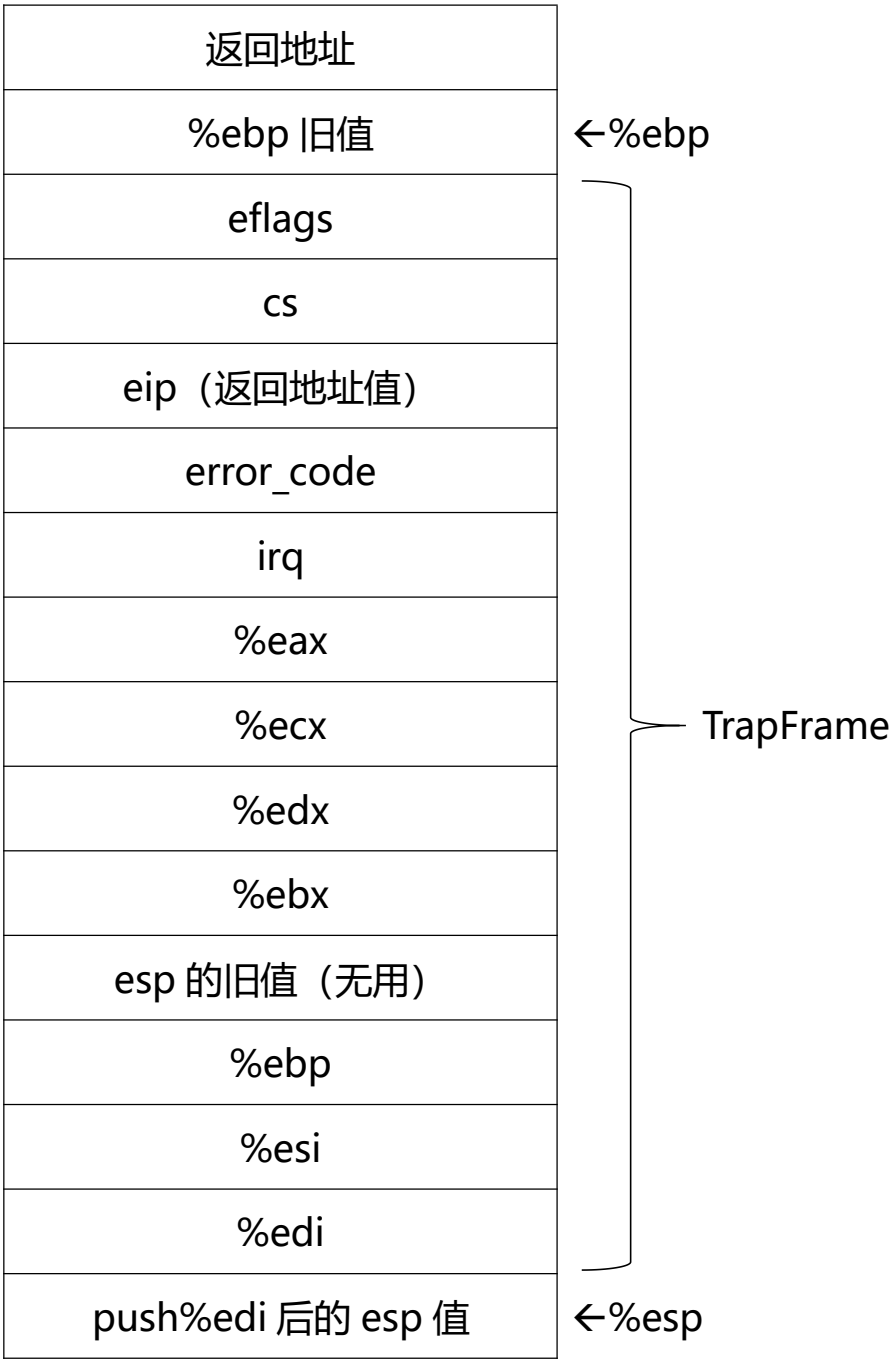
1. 详细描述从测试用例中的 int $0x80 开始一直到 HIT_GOOD_TRAP 为止的详细的系统行为（完整描述控制的转移过程，即相关函数的调用和关键参数传递过程)， 可以通过文字或画图的方式来完成。

   答：首先，测试用例中出现 int 指令，系统通过 cpu 轮询找到并调用相应的 int 函数，并将读到的 8 位立即数 “0x80” 传给 raise_sw_intr 函数并调用之。接着，raise_sw_intr 函数先将 pc 修改为下条指令的位置，再将刚刚传入的 0x80 传给 raise_intr 函数并调用之。然后，在 raise_intr 函数中，一系列参数进栈，并将 eip 值修改，进入内核态，调用 vecsys()系统函数，接着该函数调用 asm_do_irq 函数（为了执行 pusha 指令)，asm_do_irq 接着调用 irq_handle 函数,接收的形参是 Trapframe 结构的首地址(指针),通过将 esp 压栈得以保存，然后通过该结构调用 sys_call 函数，sys_call 函数又通过该结构中 eax 的值调用 sys_write 函数，sys_write 函数调用 fs_write 函数，fs_wirte 函数调用 0x82 指令，输出 “Hello, world!” 字样，然后返回用户态，输出 “HIT_GOOD_TRAP”。



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。

答： push %esp 的作用为把 Trapframe 结构的首地址压栈，为了方便 irq_handle 函数调用结构的指针参数（即首地址)。



## §4-1.3.2 响应时钟中断通过自陷实现系统调用

1. 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。

   答：NEMU：在开启时钟中断之后，NEMU 会初始化 cpu 的中断引脚，并且会在每次执行指令后根据中断引脚和 IF 的情况判断要不要进行时钟中断处理，这个过程是通过将时钟中断号传给上一个任务中的 raise_intr()函数实现的。其余部分相同。Kernel：在开启时钟中断后，Kernel 每 10ms 就会调用一次 irq_handle()函数，用来处理相应的时钟中断事件，而 irq_handle()函数的触发过程和上一个任务中通过自陷指令触发的类似。故两者结合起来的不同之处为：1.时钟中断需要开启线程以便对调用号和中断允许进行初始化，后者直接在指令中进行；2.系统调用号不同，前者是 0x80，后者是 0x20(0+32)；3.引发条件不一样，前者是自陷指令引起的系统调用（不可屏蔽），后者是每十毫秒调用一次的时钟中断函数引起的系统调用。相同之处：1.两者都需要进行用户态到内核态的准备工作，即把用户进程的相关数据放入用户栈中；2.两者都会调用 irq_handle 函数。

# PA4.2

1. 注册监听键盘事件是怎么完成的?

   答：echo 中的 main 函数首先调用了一条增加请求中断句柄的函数，该函数执行了陷阱指令，并给各个寄存器赋值，eax=0 代表调用增加请求中断句柄，ebx 为调用句柄号，ecx 为相应的处理函数指针，这里为键盘事件处理函数。

该处理函数会在相应的时刻读取键盘扫描码，然后交给扫描码翻译程序翻译，再将翻译出来的字符打印在屏幕上。main 函数中一直循环执行的 hlt 指令是为了让 cpu 处于休眠，直到下一个键盘中断信号的到来。

2. 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。

   答：键盘按下一个键后，hlt 指令监测到新的键盘中断信号（由 cpu 的中断引脚发出），引发硬件中断，并将该键盘扫描码交给键盘事件处理函数处理，然后键盘事件处理函数通过扫描码翻译程序翻译出扫描码对应的大写英文字符，并通过封装过的 printc 函数打印到屏幕上去。