pa4-1

1.

执行int $0x80时, 将0x80作为参数传递给raise\_sw\_intr().

raise\_sw\_intr()函数修改eip并将0x80作为参数传递给raise\_intr().

raise\_intr()函数保存当前现场, 即将eflags, cs的可见部分和eip的值压栈, 并根据中断号0x80读取中断向量, 修改cs和eip的值, 进行跳转.

跳转后执行asm\_do\_irq中的汇编代码, 将通用寄存器和esp压栈作为irq\_handle()的参数, esp的值就是指向TrapFrame的指针tf.

irq\_handle()函数根据中断号0x80调用相应的do\_syscall(), 参数依然是指针tf.

do\_syscall()函数根据eax调用相应的sys\_write(), 参数依然是指针tf.

sys\_write()函数调用fs\_write()函数, 参数依次是ebx, ecx, edx的值.

fs\_write()函数将待打印字符串显示在标准输出设备后逐层返回至asm\_do\_irq.

asm\_do\_irq之后的代码将通用寄存器出栈, 并调用iret()函数.

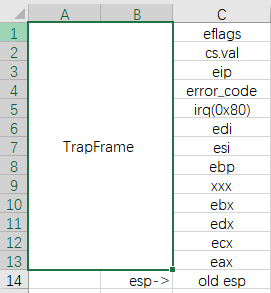
iret()函数恢复现场, 即将栈中内容出栈, 并填充至eflags, cs和eip.

之后返回至main函数, 执行HIT\_GOOD\_TRAP.

2.

push %esp起的作用：

给函数irq\_handle()传递所需参数TrapFrame \*tf.



3.

不同:

响应时钟中断在每次执行完指令后都要进行检查是否有中断到来.

系统调用在执行到int指令时才会陷入内核态执行.

相同:

时钟中断和系统调用都需要从用户态转入内核态执行.

pa4-2

1.

nemu通过NEMU\_SDL\_Thread()捕获键盘事件, 并将捕获到的扫描码作为参数给模拟键盘函数, 此函数通过中断方式调用kernel的中断处理程序.

kernel的中断处理程序查找是否有应用程序注册了对键盘事件的响应. 若有, 则通过调用注册的响应函数通知应用程序, 在应用程序的键盘响应函数中使用in指令从端口读取扫描码完成数据交换.

2.

从键盘按下一个键后, 通过pa4-2中第一题所述方式得到捕获的数据.

将该数据作为参数, 通过系统调用进行输出, 系统调用的流程见pa4-1第一题中所述内容.