计算机系统基础

实验报告

PA 4

计算机科学与技术系
191220008 陈南曈

4-1：

1.1、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。

当 cpu 执行到 int $0x80 指令时，将中断号 0x80 传递给函数 raise_intr，函数 raise_intr 通过访问寄存器 idtr 获得中断描述符表，再以中断号 0x80 为下标访问中断描述符表得到门描述符，保存现场后将 eip 指向门描述符对应的地址。然后系统就会执行异常所对应的异常处理程序。

vecsys 函数将中断号 0x80 传递给 asm_do_irq，asm_do_irq 将由硬件保存的 eip, cs, eflags 和 push 的 error_code, id 以及 pusha 保存的通用寄存器组成结构体 TrapFrame，并把指针（即首地址 esp）传递给函数 irq_handle，irq_handle 调用对应的系统函数，完成输出字符串操作。

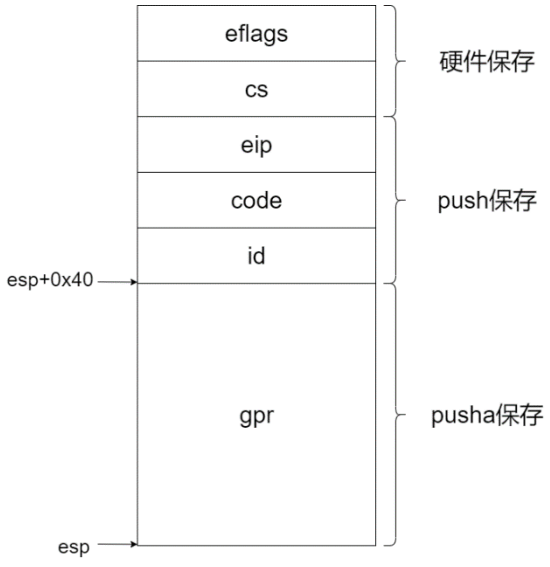
之后，通过 popa 和 iret 指令恢复栈帧和保存的现场，然后就返回之前的用户程序的下一条指令，继续执行接下来的指令，直到 HIT GOOD TRAP。

1.2、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。

push %esp 实际上是将 esp 中的内容，即 Trapframe 的首地址压入栈中，接着 esp-=4，确保其指向栈顶。

执行该指令前的栈帧如下图：



2.1、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。

首先，raise_intr 做的是把 EIP 指向 kernel 代码，真正处理时钟中断是在 kernel 代码， raise 函数的作用就是把操作系统喊过来，在下面这个地方：有一个函数指针指向的函数会处理，irq_handle.c 里面创建了一个关于系统调用的链表数组，链表结构，但是每一个元素对应一个系统调用号的数组。

相同之处：整个过程就是 push 现场之后，在 irq_handle.c 里面看是哪个系统调用；都是通过 int $0x80 这样的形式陷入 kernel 层次的代码（只是模拟层面是用 C 语言的 if-else，本质上是一样的），由 kernel 来处理中断。

不同之处：时钟中断某种意义是异步发生的。通过向 intr 发个信号，将异常号放在 PIC 总线上，触发中断。在当前指令被 CPU 处理完之后，CPU 发现 intr 为 1，就读取异常号，调用中断处理程序。而之前的 int 0x80 是产生的中断是执行当前指令的结果，是同步发生的。