161220097 戚赟 计算机科学与技术系

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；

用户进程执行 int $0x80 之后, CPU将会保存现场, 查阅kenrel设置好的IDT, 跳转到入口函数vecsys() , 压入错误码和异常号 #irq , 跳转到 raise\_intr() . 在 raise\_intr() 中, 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, #irq , 以及硬件保存的EFLAGS, CS, EIP都将会被保存（运用的指令是PUSHA）. 之后, kernel就可以随意使用通用寄存器了. 接下来代码将会把当前的 %esp 压栈, 并通过raisr\_intr()这个函数来调用C函数 irq\_handle() .irq\_handle() 将会根据 #irq 确定异常事件的类型, 从而进行不同的处理. 用户进程执行的int $0x80 会被识别成系统调用请求, 于是kernel会调用 do\_syscall() 对相应的请求进行处理. do\_syscall() 将根据用户进程之前设置好的系统调用号和系统调用参数进行处理. 利用打印的话，利用nemutrap来进行屏幕的输出。接下来的事情就是恢复用户进程的现场, kernel将根据之前保存的内容, 恢复用户进程的通用寄存器并直接弹出一些不再需要的信息, 最后通过 iret 指令恢复用户进程的EIP, CS, EFLAGS. 用户进程可以通过 %eax寄存器获得系统调用的返回值, 进而得知系统调用执行的结果 .

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起的作用就是将中断之前的栈的%esp的位置保存进入，然后再用栈保存各类寄存器。

|  |
| --- |
| 旧的EFLAGS |
| 旧的CS |
| 旧的EIP |
| …… |
| 旧的 EAX |
| 旧的 ECX |
| 旧的 EDX |
| 旧的 EBX |
| 旧的 ESP |
| 旧的 EBP |
| 旧的 ESI |
| 旧的 EDI |

……

%esp + 4

%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。

不同的方式是系统调用过程是系统的自陷模式，不需要任何外部的信号。而相应时钟中断则是这样，cpu里有一个INTR，每次时钟以固定的HZ进行相应检测，当检测到INTR的为1，那么就调用相应的处理结果，这个是由外设控制的，且不同点是随机性而且只能有一个触发中断的事件发生。

相同点是，都会调用奥SYS\_\_Call()来进行对相应的中断号的处理。从irq\_handle()往后的调用过程和一开始的调用过程基本类似。

4.

1. 注册监听键盘事件是怎么完成的？

当按下一个键的时候, 键盘控制器将会发送该键的通码(make code); 当释放一个键的时候, 键盘控制器将会发送该键的断码(break code), 其中断码的值为通码的值+0x80.

当设备需要发出硬件中断时, 就会调用raise\_intr() , 代码会根据当前的中断请求状态选择一个优先级最高的中断, 并生成相应的中断号, 把中断号记录在intr\_NO 变量中, 然后把CPU的INTR引脚置为高电平, 通知CPU有硬件中断到来. 中断控制器收到CPU的确认后会更新中断请求的状态, 清除刚才发送给CPU的中断请求. 每当用户敲下/释放按键时, 将会把键盘扫描码放入数据寄存器, 然后发起键盘中断, CPU收到中断后, 可以通过端口I/O访问数据寄存器, 获得键盘扫描码 。从而完成了对键盘的注册监听。

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。

按下键盘后，当设备需要发出硬件中断时, ,CPU将会保存现场, 查阅kenrel设置好的IDT, 压入错误码和异常号 #irq , 就会调用raise\_intr(). 在raise\_intr() 中, 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, #irq , 以及硬件保存的EFLAGS, CS, EIP都将会被保存（运用的指令是PUSHA）. 之后, kernel就可以随意使用通用寄存器了. 接下来代码将会把当前的 %esp 压栈, 并通过raisr\_intr()这个函数来调用C函数 irq\_handle() .irq\_handle() 将会根据 #irq 确定异常事件的类型, 从而进行不同的处理. kernel会调用 do\_syscall() 对相应的请求进行处理. do\_syscall() 将根据用户进程之前设置好的系统调用号和系统调用参数进行处理.接下来的事情就是恢复用户进程的现场, kernel将根据之前保存的内容, 恢复用户进程的通用寄存器并直接弹出一些不再需要的信息, 最后通过 iret 指令恢复用户进程的EIP, CS, EFLAGS. 用户进程可以通过 %eax寄存器获得系统调用的返回值, 进而得知系统调用执行的结果 .