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

```
.globl vecsys; vecsys:  pushl $0;  pushl $0x80; jmp asm_do_irq

.globl asm_do_irq
.extern irq_handle

asm_do_irq:
```

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

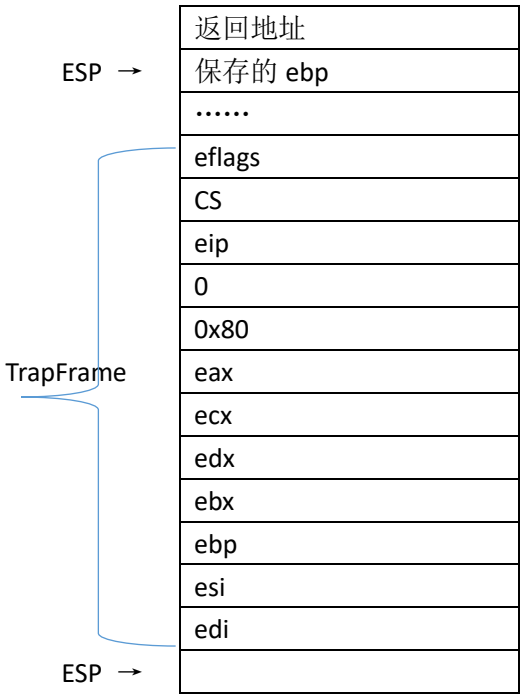
此后，cpu.eip 回到 int 指令的后一条指令，便继续执行 hello-inline 的代码

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：



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。

```
#if defined(HAS_DEVICE_TIMER) || defined(HAS_DEVICE_VGA) || defined(HAS_DEVICE_KEYBOARD)
    /* Initialize SDL for timer displayer and keyboard*/
    init_sdl();
#endif
```

开启 HAD_DEVICE_TIMER 后，调用 init_sdl()函数，创建一个 SDL 线程

```
void init_sdl() {
    if(!initialized) {
#ifdef HAS_DEVICE_VGA
        init_vga();
#endif
        SDL_Thread *thread;
        // int threadReturnValue;
        // Simply create a thread
        thread = SDL_CreateThread(NEMU_SDL_Thread, (void *)NULL);

        if (NULL == thread) {
            printf("\nSDL_CreateThread failed: %s\n", SDL_GetError());
            assert(0);
        }
    }
}
#endif
```

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

```
#ifdef HAS_DEVICE_TIMER
        timer_intr();
```

```
void i8259_raise_intr(uint8_t irq_no) {
    intr_no = irq_no + IRQ_BASE;
#ifdef HAS_DEVICE_TIMER
    cpu.intr = 1;
#endif
}
```

```
void timer_intr() {
    if(nemu_state == NEMU_RUN) {
        i8259_raise_intr(TIMER_IRQ);
    }
}
```

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。

```
#ifdef HAS_DEVICE_TIMER
        //printf("\nPlease call do_intr() here\n");
        //assert(0);
        do_intr();
```

```
static void do_intr() {
    // check for interrupt
    if(cpu.intr && cpu.eflags.IF) {
        // get interrupt number
        uint8_t intr_no = i8259_query_intr_no(); // get interrupt number
        assert(intr_no != I8259_NO_INTR);
        i8259_ack_intr(); // tell the PIC interrupt info received
        raise_intr(intr_no); // raise intrrupt to turn into kernel handler
    }
}
```

4. 注册监听键盘事件是怎么完成的？

```
int main() {
    // register for keyboard events
    add_irq_handler(1, keyboard_event_handler);
    while(1) asm volatile("hlt");
    return 0;
}
```

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。

```
void
add_irq_handle(int irq, void (*func)(void) ) {
    assert(irq < NR_HARD_INTR);
    assert(handle_count <= NR_IRQ_HANDLE);

    struct IRQ_t *ptr;
    ptr = &handle_pool[handle_count ++]; /* get a free handler */
    ptr->routine = func;
    ptr->next = handles[irq]; /* insert into the linked list */
    handles[irq] = ptr;
}
```

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。

```
int main() {
    // register for keyboard events
    add_irq_handler(1, keyboard_event_handler);
    while(1) asm volatile("hlt");
    return 0;
}
```

```
make_instr_func(hlt) {
    SDL_Thread *thread;
    print_asm_0("hlt", "", 1);
    thread = SDL_CreateThread(HLT_Thread, (void *)NULL);

    if (NULL == thread) {
        printf("\nSDL_CreateThread for HLT failed: %s\n", SDL_GetError());
    } else {
        SDL_WaitThread(thread, NULL);
    }
```

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

```
#ifdef HAS_DEVICE_KEYBOARD
            else if (e.type == SDL_KEYDOWN){
                keyboard_down(e.key.keysym.sym);
            } else if (e.type == SDL_KEYUP) {
                keyboard_up(e.key.keysym.sym);
            }
```

```
// called by the nemu_sdl_thread on detecting a key down event
void keyboard_down(uint32_t sym) {
    // put the scan code into the buffer
    scan_code_buf = sym2scancode[sym >> 8][sym & 0xff];
    // issue an iterrupt
    i8259_raise_intr(KEYBOARD_IRQ);
    // maybe the kernel will be interested and come to read on the data port
}

// called by the nemu_sdl_thread on detecting a key up event
void keyboard_up(uint32_t sym) {
    // put the scan code into the buffer
    scan_code_buf = sym2scancode[sym >> 8][sym & 0xff] | 0x80;
    // issue an iterrupt
    i8259_raise_intr(KEYBOARD_IRQ);
    // maybe the kernel will be interested and come to read on the data port
}
```

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

```
// the keyboard event handler, called when an keyboard interrupt is fired
void keyboard_event_handler() {

    uint8_t key_pressed = in_byte(0x60);

    // translate scan code to ASCII
    char c = translate_key(key_pressed);
    if(c > 0) {
        // can you now fully understand Fig. 8.3 on pg. 317 of the text book?
        printc(c);
    }

}
```