PA4-2 实验报告

191220163 计算机科学与技术系 张木子苗

针对echo测试用例，在实验报告中，结合代码详细描述：

1. 注册监听键盘事件是怎么完成的?

   （1）echo.c 的 main 函数调用了 add_irq_handler 函数，该函数通过 int 0x80 指令调用系统函数 add_irq_handle， 将对应的处理程序（函数指针）添加到 kernel 的异常处理程序中。

   （2）对键盘展开模拟时，键盘事件首先在 nemu/src/device/sdl.c 中由 NEMU_SDL_Thread() 线程捕获。NEMU 捕获两类事件：键盘按下和抬起。

```
// called by do_keyboard() 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 do_keyboard() 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
}
```

   当检测到相应事件后，将对应键的扫描码作为参数传送给 keyboard.c 中的模拟键盘函数。模拟键盘缓存扫描码，并通过中断请求的方式通知 CPU 有按键或抬起的事件，键盘的中断请求号为 1。

   （3）CPU 收到中断请求后调用 Kernel 的中断响应程序。 在响应程序中，Kernel 会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的方式来通知应用程序。此时在应用程序的键盘响应函数中，可以通过 in 指令从键盘的数据端口读取扫描码完成数据交换。

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。

   （1）按下键盘按键时会产生一个外部中断（键盘向 i8259 发送信号）

   （2）在每条指令执行后，cpu 调用 do_intr() 函数，检测是否有中断发生，如果有 `(cpu.intr && cpu.eflags.IF)`，则利用 i8259_query_intr_no() 函数查询中断号，转 raise_intr() 处理。

   （3）函数 raise_intr 通过访问寄存器 idtr 获得中断描述符表首地址，再以中断号为下标访问中断描述符表得到门描述符，保存 eflags 和断点（cs，eip）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。

```
.globl irq1;     irq1:  pushl $0;  pushl $1001; jmp asm_do_irq
```

   （4）键盘中断对应的传参函数是 irq1 函数。irq1 函数将硬件错误码 0 和中断号 1001 压栈，转到 asm_do_irq，asm_do_irq 将先用 pusha 保存通用寄存器，再将由硬件保存的 eip, cs, eflags 和 push 的 error_code, irq 以及 pusha 保存的通用寄存器组成结构体 TrapFrame，并把结构体首地址 （即%esp）传递给函数 irq_handle，irq_handle 根据中断号1001决定进行键盘中断响应，之

后会跳转到 echo 注册的处理程序执行。处理程序通过 in 指令从键盘 io 读入数据，并将其转换为 ascii 码，再通过系统调用将 ascii 码输出。

```
asm_do_irq:
    pushal

    pushl %esp      # ???
    call irq_handle

    addl $4, %esp
    popal
    addl $8, %esp
    iret
```

（5）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。