* **实验报告要求**

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？

我们对于 echo.c 中的函数进行分析：

```
// read a byte from the port
uint8_t in_byte(uint16_t port)
{
    uint8_t data;
    asm volatile("in %1, %0"
                 : "=a"(data)
                 : "d"(port));
    return data;
}
```

从端口读入一个字节的数据。

```
// register a handle of interrupt request in the Kernel
void add_irq_handler(int irq, void *handler)
{
    // refer to kernel/src/syscall/do_syscall.c to understand what has happened
    asm volatile("int $0x80"
                 :
                 : "a"(0), "b"(irq), "c"(handler));
}
```

注册监听事件：把 Keyboard_event_handler 和键盘输入的中断异常处理程序进行绑定。

```
// 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);
    }
}
```

键盘中断处理程序。

接下来我们来看执行的过程：

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

执行 int 0x80，于是我们按照提示找到 kernel/src/syscall/do_syscall.c：

```
void do_syscall(TrapFrame *tf)
{
    switch (tf->eax)
    {
    case 0:
        cli();
        add_irq_handle(tf->ebx, (void *)tf->ecx);
        sti();
        break;
    case SYS_brk:
        sys_brk(tf);
```

由 eax=0，知道进一步调用 add_irq_handle，注册监听事件，将 Keyboard_event_handler 和键盘输入的中断异常处理程序进行绑定。

```
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;
}
```

插入到空的处理函数位置。

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。

   此时，我们已经注册了键盘监听事件，将键盘中断与键盘中断处理程序 Keyboard_event_handler 绑定。现在，我们按下一个键，操作系统检测到中断（具体详见 pa4-1），调用中断处理程序 keyboard_event_handler，首先执行指令 in (in_byte 函数)，in 指令中调用 pio_read，把读取的结果存储在 eax 中，返回搭配 in_byte 函数输出。传入到 translate_key 中将键码转化为对应的 ascii 码，再进一步调用 printc，通过 printc 调用 writec，而 writec:

   ```
   "int $0x80" : : "a"(SYS_write), "b"(fd), "c"(&c), "d"(1)
   ```

   由 `int $0x80` 进一步调用 `sys_write`，`sys_write` 调用 `fs_write`，`fs_write` 进一步调用，一直调用到 `out_byte` 函数，通过 `out` 指令进行输出。
   等待下一次中断到来。