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？

注册监听键盘事件的测试样例是 echo，即观察 echo.c 中代码：

```
53 int main()
54 {
55     // register for keyboard events
56     add_irq_handler(1, keyboard_event_handler);
57     while (1)
58         asm volatile("hlt");
59     return 0;
60 }
61
```

调用了 add_irq_handler，其实现如下：

```
15 // register a handle of interrupt request in the Kernel
16 void add_irq_handler(int irq, void *handler)
17 {
18     // refer to kernel/src/syscall/do_syscall.c to understand what has happened
19     asm volatile("int $0x80"
20                  :
21                  : "a"(0), "b"(irq), "c"(handler));
22 }
23
```

可见该函数的功能为注册监听事件，即将 handler 与中断异常处理程序绑定
根据 int $0x80 可知为系统调用，又此时 eax 为 0，找到 do_syscall 代码：

```
46 void do_syscall(TrapFrame *tf)
47 {
48     switch (tf->eax)
49     {
50     case 0:
51         cli();
52         add_irq_handle(tf->ebx, (void *)tf->ecx);
53         sti();
54         break;
```

其中调用了 add_irq_handle，找到这个函数的定义：

```
21 void add_irq_handle(int irq, void (*func)(void))
22 {
23     assert(irq < NR_HARD_INTR);
24     assert(handle_count <= NR_IRQ_HANDLE);
25
26     struct IRQ_t *ptr;
27     ptr = &handle_pool[handle_count++]; /* get a free handler */
28     ptr->routine = func;
29     ptr->next = handles[irq]; /* insert into the linked list */
30     handles[irq] = ptr;
31 }
```

在 handle_pool 里找到空的位置插入，此处 func 为 keyboard_event_handler：

```
38 // the keyboard event handler, called when an keyboard interrupt is fired
39 void keyboard_event_handler()
40 {
41
42     uint8_t key_pressed = in_byte(0x60);
43
44     // translate scan code to ASCII
45     char c = translate_key(key_pressed);
46     if (c > 0)
47     {
48         // can you now fully understand Fig. 8.3 on pg. 317 of the text book?
49         printc(c);
50     }
51 }
52
```

可见 keyboard_event_handler 调用 in_byte 来获取按键输入的键码
然后通过 translate_key 来将输入的键码“翻译”为 ascii 码：

```
// 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;
}

```

观察 in_byte 可见它的功能是从端口读入一个字节的数据

```
char translate_key(int scan_code)
{
    int i;
    for (i = 0; i < 26; i++)
    {
        if (letter_code[i] == scan_code)
        {
            return i + 0x41;
        }
    }
    return 0;
}
```

可见 translate_key 的作用是依次检查键盘输入是否在 26 个字母之间
若在 26 个字母中，则返回对应字母的 ascii 码，否则返回值为 0

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？

按下键盘出现异常，系统调用函数与 keyboard_event_handler 绑定，其中调用 in_byte，该函数调用汇编指令 in，in 指令调用了 pio_read，读取的结果返回值储存在 data 中，继续调用 translate_key 转化成 ascii 码，通过 printc 调用调用 writec 输出字符：

```
void writec(int fd, char c)
{
    asm volatile("int $0x80"
                 :
                 : "a"(SYS_write), "b"(fd), "c"(&c), "d"(1));
}

void printc(char c)
{
    writec(1, c);
}

```

可见这里又有一个系统调用，经 do_syscall 调用 sys_write：

```
    case SYS_write:
        sys_write(tf);
        break;
```

```
static void sys_write(TrapFrame *tf)
{
    tf->eax = fs_write(tf->ebx, (void *)tf->ecx, tf->edx);
}

```

然后 sys_write 调用 fs_write，传入输入的字符指针和长度：

```
51 size_t fs_write(int fd, void *buf, size_t len)
52 {
53     assert(fd <= 2);
54 #ifdef HAS_DEVICE_SERIAL
55     int i;
56     extern void serial_printc(char);
57     for (i = 0; i < len; i++)
58     {
59         serial_printc(((char *)buf)[i]);
60     }
61 #else
62     asm volatile(".byte 0x82"
63                  : "=a"(len)
64                  : "a"(4), "b"(fd), "c"(buf), "d"(len));
65 #endif
66
67     return len;
68 }
69
```

可见调用 serial_printc，每次输出一个字符：

```
14 void serial_printc(char ch)
15 {
16     while (!serial_idle())
17         ; // wait untile serial is idle
18     // print 'ch' via out instruction here
19     out_byte(SERIAL_PORT, ch);
20     //HIT_BAD_TRAP;
21 }
22
```

```
26 static inline void
27 out_byte(uint16_t port, uint8_t data)
28 {
29     asm volatile("out %%al, %%dx"
30                  :
31                  : "a"(data), "d"(port));
32 }
33
```

其中又调用了 out_byte，调用汇编指令 out，利用 pio_write 实现输出