Lab2 实验报告

陈永恒 151220012

实验要求

1. 填写中断描述符表，增加系统调用处理例程，或者修改通用中断处理函数，发现中断是系统调用时跳转到系统调用处理函数；
2. 将处理硬件（串口、显存、键盘）的代码移入对应的系统调用处理函数中；
3. 封装系统调用接口（系统原语），用原语替换游戏代码中对应功能的部分代码；

实验进度

所有要求已经实现，具体如下：

1. 填写中断描述符表，增加系统调用处理例程

```
79      set_trap(idt + 13, SEG_KERNEL_CODE, (uint32_t)vec13, DPL_KERNEL);
80      set_trap(idt + 0x80, SEG_KERNEL_CODE, (uint32_t)vecsys, DPL_USER);
```

```
16  .globl vec13;   vec13:  pushl $0;  pushl   $13; jmp asm_do_irq
17  .globl vec14;   vec14:  pushl $0;  pushl   $14; jmp asm_do_irq
18
19  .globl vecsys; vecsys:  pushl $0;  pushl $0x80; jmp asm_do_irq
20
21  .globl irq0;     irq0:  pushl $0;  pushl $1000; jmp asm_do_irq
22  .globl irq1;     irq1:  pushl $0;  pushl $1001; jmp asm_do_irq
23  .globl irq14;   irq14:  pushl $0;  pushl $1014; jmp asm_do_irq
```

```
36          case 4000:
37              tf->eax = get_tick();
38          break;
39          case 4001:
40              tf->eax = get_key(tf->ebx);
41          break;
```

2. 区分了kernel和游戏文件，游戏由kernel加载并切换：

```
    struct ProgramHeader *ph, *eph;
    unsigned char* pa, *i;

    readseg((unsigned char*)elf, 8*SECTSIZE, 102400);

    printf("Magic Assertion: %x\n", (elf->magic == 0x464C457FU));


    ph = (struct ProgramHeader*)((char *)elf + elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph ++) {
      pa = (unsigned char*)ph->paddr;
      readseg(pa, ph->filesz, 102400+ph->off);
      for (i = pa + ph->filesz; i < pa + ph->memsz; *i ++ = 0);
    }
    enable_interrupt();
   ((void(*)(void))elf->entry)();
    printf("%s\n", "Never return otherwise you are fucked!");
    while(1){
    }
    return 1;
};
```

且游戏的处理硬件（串口、显存、键盘）的代码移入了对应的系统调用处理函数

3. 封装了printf()等系统函数;
4. 实现分段机制

```
static void
set_segment(SegDesc *ptr, uint32_t pl, uint32_t type) {
    ptr->limit_15_0  = 0xFFFF;
    ptr->base_15_0   = 0x0;
    ptr->base_23_16  = 0x0;
    ptr->type = type;
    ptr->segment_type = 1;
    ptr->privilege_level = pl;
    ptr->present = 1;
    ptr->limit_19_16 = 0xF;
    ptr->soft_use = 0;
    ptr->operation_size = 0;
    ptr->pad0 = 1;
    ptr->granularity = 1;
    ptr->base_31_24  = 0x0;
}
```

```
void
init_segment(void) {
    memset(gdt, 0, sizeof(gdt));
    set_segment(&gdt[SEG_KERNEL_CODE], DPL_KERNEL, SEG_EXECUTABLE | SEG_READABLE);
    set_segment(&gdt[SEG_KERNEL_DATA], DPL_KERNEL, SEG_WRITABLE );
    set_segment(&gdt[SEG_USER_CODE], DPL_USER, SEG_EXECUTABLE | SEG_READABLE);
    set_segment(&gdt[SEG_USER_DATA], DPL_USER, SEG_WRITABLE );


    //set_segment(&gdt[SEG_TSS], DPL_USER, SEG_EXECUTABLE | SEG_READABLE );

    write_gdtr(gdt, sizeof(gdt));

    set_tss(&gdt[SEG_TSS]);
    write_tr( SELECTOR_USER(SEG_TSS) );
}
```

git log 部分截图:

```
commit 457f4c8e092d88c50051c2e44b7e5c715e64a2ca
Author: 151220012-chango chen <changochen1@gmail.com>
Date:   Sun Apr 9 22:38:30 2017 +0800

    Finished all required function

commit d6842d687648cb73ed6aa395aff21adb73a2258d
Author: 151220012-chango chen <changochen1@gmail.com>
Date:   Sun Apr 9 13:25:20 2017 +0800

    add some system call

commit 0dd1df93491bf9072633bf183f3c1e96864e19db
Author: 151220012-chango chen <changochen1@gmail.com>
Date:   Sun Apr 9 09:08:35 2017 +0800

    add printf

commit 4dd9b16e4fb11aec79585faabbc74d1868c3dea8
Author: 151220012-chango chen <changochen1@gmail.com>
Date:   Sun Apr 9 08:58:42 2017 +0800

    add system call
```

实验中遇到的问题及解决方案

1. 在kernel加载游戏文件时，会出现越界错误，调试调了很久，发现是没有注册时间和键盘中断所导致

```
44          init_intr();
45          //set_keyboard_intr_handler(nothing2);
46          //set_timer_intr_handler(nothing);
47
```

> 解决方法:在加载游戏文件时就必须把其注册，否则时钟信号没有被处理，发生错误

2. kernel和游戏的分离，使得我们不能直接在游戏中使用处理时钟信号的函数，所以我决定加一个计时器:

```
1   #include<lib/common.h>
2   unsigned int ticks=0;
3
4   void do_timer(){
5       ticks++;
6   }
7
8
9   unsigned int get_tick(){
10      return ticks;
11  }
```

在每个时钟中断时调用do_timer()计数．后来在游戏中直接调用get_tick()的时候，发现得到的计时器的值总是0，然而我发现在kernel中调用get_tick()的时候，发现得到的计时器的值是正常的．后来突然想起这其实已经是两个独立的进程了,不能这样直接调用.于是就增加了一个系统调用来取得kernel中的计数器的值．