中山大学数据科学与计算机学院
操作系统实验课程

实 验 报 告

教　　师　凌应标

学　　号　17341035

姓　　名　傅畅

实验名称　实验六（二状态多进程模型）

实 验 六

二 状 态 多 进 程 模 型

姓名：傅畅 **学号：**17341038

邮箱：fuch8@mail2.sysu.edu.cn

实验时间：周五（3-4 节）

目录

一、 实验要求

• 在 c 程序中定义进程表，进程数量为 4 个。

• 内核一次性加载 4 个用户程序运行时，采用时间片轮转调度进程运行，用户程序的输出各占 1/4 屏幕区域，信息输出有动感，以便观察程序是否在执行。

• 在原型中保证原有的系统调用服务可用。再编写 1 个用户程序，展示系统调用服务还能工作。

二、 实验配置

（一）实验支撑环境

• 硬件：个人计算机

• 主机操作系统：Linux 4.18.0-16-generic 17-Ubuntu

• 虚拟机软件: Bochs 2.6.9

三、 局部程序数据结构设计

（一）局部段描述符

为了和内核的数据代码段区分开来，需要为局部用户程序的代码申请独立的空间，并安装独立的描述符。区别与 GDT，每个用户程序有一张自己的段描述符表 LDT。它的格式和 GDT 类似，只不过如果要访问 LDT 中的描述符，选择子的 TI 位必须为 1，RPL 需要视情况而定，一般为 CPL。

用户之间的的 LDT 也彼此独立，所以整个程序有多个 LDT 空间。由于 LDT 本身同样是一段内存，也是一个段，所以它也有个描述符描述它，这个 LDT 的描述符放在 GDT 中。

对应这个表述符也会有一个选择子，LDTR 装载的就是这样一个选择子, 加载制定的 LDT 空间需要指令:lldt。当然也可以在任务切换时执行。

（二）**TSS**

在一个多任务环境中，当任务发生切换时，必须保存现场（比如通用寄存器，段寄存器，栈指针等）。为了保存被切换任务的状态，并且在下次执行它时恢复现场，每个任务都应当有一片内存区域，专门用于保存现场信息，x86 默认支持这样一种任务状态段（Task State Segment）的格式。

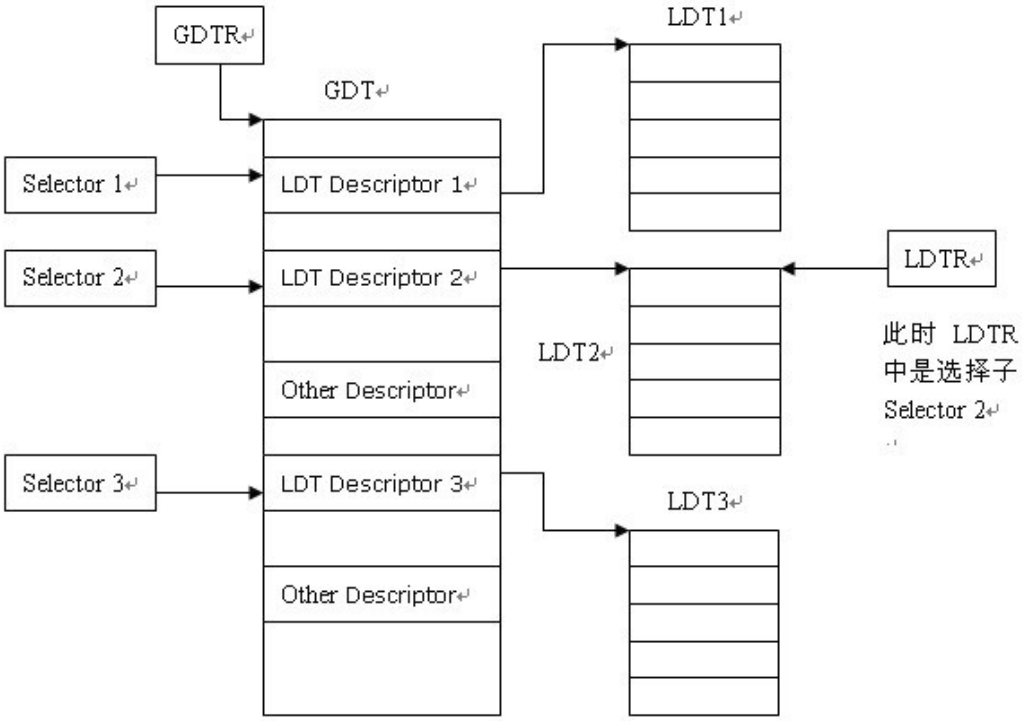
GDTR
GDT
LDT1
Selector 1
LDT Descriptor 1
Selector 2
LDT Descriptor 2
LDTR
此时 LDTR
中是选择子
Selector 2
LDT2
Other Descriptor
Selector 3
LDT Descriptor 3
LDT3
Other Descriptor
图五

图 1: LDT 布局

在创建一个任务的时候，我们要为这个任务创建 TSS 并填写其中的一些字段。

• 前一任务链接（TSS Back Link）：TSS 内偏移 0 处是前一个任务的 TSS 描述符的选择子。当 Call 指令、中断或者异常造成任务切换，处理器会把旧任务的 TSS 选择子复制到新任务的 TSS 的 Back Link 字段中

• SS0，SS1，SS2 和 ESP0，ESP1，ESP2 分别是 0,1,2 特权级堆栈的选择子和栈顶指针。这些内容应当由任务的创建者填写，且属于填写后一般不变的静态字段。

• CR3, 指用户程序的页目录物理地址。供页表切换时使用。

• 偏移为 0x20 0x5C 的区域是处理器各个寄存器的快照部分，用于任务切换时保存现场。在一个多任务环境中，每次创建一个任务，内核至少要填写 EIP,EFLAGS,ESP,CS,DS,SS,ES,和 GS。

• LDT 选择子是当前任务的 LDT 选择子，由内核填写，以指向当前任务的 LDT。该信息由处理器在任务切换时使用，在任务运行期间保持不变。

• T（Debug Trap）位用于软件调试。在多任务环境中，如果 T=1，则每次切换到该任务的时候，会引发一个调试异常中断（int 1）.

• I/O 位图基地址用来决定当前的任务是否可以访问特定的硬件端口。

和 LDT 一样，必须为每个 TSS 在 GDT 中创建对应的描述符。

任务是不可重入的。就是说在多任务环境中，如果一个任务是当前任务，那么它可以切换到其他任务，但是不能从自己切换到自己。在 TSS 描述符中设置 B 位，并由处理器固件进行管理，可以防止这种情况的发生。

（三）TCB

尽管不是处理器的要求，为了便于内核程序组织，我们额外还设计了 TCB 的数据结构。用于简要记录 LDT，TSS 在内核页表映射下的位置。

为了表示程序运行状态，增加 State 用 +-1 表示是否正在运行（相当于 TSS 的 Busy 位）在创建一个任务，需要使用比如程序的大小，加载的位置等等，当任务执行结束，还要依据这些信息来回收任务所占用的内存空间。

（四）为高特权级使用的栈

如果在 Y 用户程序执行的过程中发生了由门发起的任务切换，那么就需要使用用户所提供的栈。毕竟这个新的任务在逻辑上还是隶属于用户程序。所以在分配内存的时候，需要为所有高于当前用户的特权级都设立一个栈。

四、 实验代码设计

（一）内存管理

比起实验五，这次加载的用户程序长度不定，内存加载逐渐变得动态了起来。为了管理低端 2MB 的内存空间，我们建立一个 bitmap 来记录哪些页已经被使用过了哪些还没有。分配内存的时候以此来遍历空闲页

```
#define page_map_len 64
u8 page_bit_map[page_map_len]=
                                {0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                                                                //低端都是土狼驻地地
                                    方
                                        0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
                                            ,
                                0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                                0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
                                0x55,0x55,0x55,0x55,0x55,0x55,0x55,0x55,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

u32 allocate_a_4k_page(){
        for (int i=0; i<page_map_len<<3; ++i){
                if ( (page_bit_map[i>>3]&(1<<(i&0x7)))==0){
                        page_bit_map[i>>3]^=1<<(i&0x7);
                        return i<<12;
```

```
17                  }
18          }
19          simple_puts("Wrong On allocated page!", (3*80<<16)+0x8);
20          return -1;
21  }
```

然后就需要实现在当前页目录下为某个线性地址分配页了。先由页目录找页表，再由页表找页。每一步不存在的页都需要进行分配。

```
1   alloc_inst_a_page:                                          ;分配一个页，并安装在当前活动的
2                                                               ;层级分页结构中
3                                                               ;void alloc(页的线性地址)
4                                                                                                   ;
                                                                                                    如果存在就不分配了
5                   push eax
6                   push ebx
7                   push esi
8
9                   ;check the exist of PageSheet
10                  mov esi, [esp+0x10]
11                  shr esi, 22
```

```
12                  shl esi, 2
13                  or esi, 0xfffff000
14                  test dword [esi], 0x01
15                  jnz .b1
16
17                  ;创建该线性地址所对应的页表
18          call allocate_a_4k_page             ;分配一个页做为页表
19          or eax,0x00000007
20          mov [esi],eax
21          .b1:
22
23                  mov esi, [esp+0x10]
24                  shr esi, 12
25                  shl esi, 2
26                  or esi, 0xffc00000
27
28                  test dword[esi], 0x01
29                  jnz .b2
30
31                  call allocate_a_4k_page
32                  or eax, 0x07
33                  mov [esi], eax
34          .b2:
35                  pop esi
36                  pop ebx
37                  pop eax
38
39                  ret
```

（二）加载用户程序

用户程序在加载的过程大概如下：

• 将当前页目录的前 2GB 清空，然后加载用户程序到 0x0 的线性地址。

• 申请 TCB，TSS 和 LDT 的地址空间，然后一边申请数据段，代码段和栈段，一边安装描述符到 LDT 和 GDT

• 将剩余的空间安装完毕，申请新的页目录，然后拷贝当前内核页目录到新的地址。

```
1  extern void Load_program(int sectors){
2          Clean_partial_PDE();
                    //清理前2GB的页目录
3          read_hard_disk_1(sectors,c_buf);
4          u32 siz = *((u32*)c_buf);
5          u32 totsec=((siz+0x0fff)>>12)<<3;
6          struct TCB *t = &tottcb[cnttcb++];
7          u32 prog_addr=0;                                          //用户
               程序的起始线性地址为0
8          for (u32 i=0; i<totsec || i<80; ++i,prog_addr+=512){   // 至少分配10页
               给代码和GCC数据段
9                  alloc_inst_a_page(prog_addr);
10                 if ( i<totsec) read_hard_disk_1(sectors+i, prog_addr);
11         }
12
13         t->TSS_bas=TSS_array; TSS_array+=0x1000; // allocate 4KB/per tss,
               TSSarray指向全局的地址空间
14         t->TSS_lim=103;
15
16         alloc_inst_a_page(t->LDT_bas=prog_addr);                  //分配
               一个页作LDT
17         t->LDT_lim=-1;prog_addr+=0x1000;
```

```
18
19          struct TSS *tssp = t->TSS_bas;
20          tssp->CS=AddDescri_2_ldt(
21                  0x00000000,0x00000009,0x00c0f800, t
22          )|0x3; // 建立代码段描述符并放入ldt中，返回段选择子,特权级设为3， 长度
                10页
23          tssp->DS=tssp->ES=tssp->FS=tssp->GS=AddDescri_2_ldt(
24                  0x00000000,0x0000001A,0x00c0f200,t
25          )|0x3;// 建立数据段描述符并放入ldt中，返回段选择子。特权级设为3 ，长度
                27页
26
27          //将数据段作为用户任务的3特权级固有堆栈
28          tssp->SS=tssp->DS;tssp->ESP=0x1b*0x1000;
                                  //能不能不减4？
29
30          //在用户任务的局部地址空间内创建0特权级堆栈，长度1页
31          alloc_inst_a_page(prog_addr);
32          tssp->SS0=AddDescri_2_ldt(
33                  prog_addr, 0x00000fff, 0x00409200,t
34          )|0x0; // ;设置选择子的特权级为0
35          prog_addr+=0x1000;
36          tssp->ESP0=0x00001000;
                      // bug, 真实地址==esp+ssbase!!!!!! esp0=0xfff!!!!而不是
                prog_addr
37
38          //在用户任务的局部地址空间内创建1特权级堆栈，长度1页
39          alloc_inst_a_page(prog_addr);
40          tssp->SS1=AddDescri_2_ldt(
41                  prog_addr, 0x00000fff, 0x0040b200,t                    // why
```

```
                ???????
    )|0x1; // ;设置选择子的特权级为1
    prog_addr+=0x1000;
    tssp->ESP1=0x00001000;

    //在用户任务的局部地址空间内创建2特权级堆栈，长度1页
    alloc_inst_a_page(prog_addr);
    tssp->SS2=AddDescri_2_ldt(
            0x0, 0x00000fff, 0x0040d200,t
    )|0x2; // ;设置选择子的特权级为2
    prog_addr+=0x1000;
    tssp->ESP2=0x00001000;

    //;在GDT中登记LDT描述符,并填写到TCB，TSS中
    t->LDT_sel=tssp->LDTsel=AddDescri_2_gdt(
            t->LDT_bas, t->LDT_lim, 0x00408200
    );

    tssp->preTSS=0x0;tssp->IOmap=t->TSS_lim;
    tssp->T=0;

    //;在GDT中登记TSS描述符,并填写到TCB,
    t->TSS_sel=AddDescri_2_gdt(
            t->TSS_bas, t->TSS_lim, 0x00408900
    );

    //总共27页，全部分配完  多分配一页？
    for (;prog_addr<(0x1C<<12); prog_addr+=0x1000)
            alloc_inst_a_page(prog_addr);
```

```
70
71
72          alloc_inst_a_page(0xffffe000);//分配一个页作页目录,由于还没有切换页表,
                对新页表的操作还需要在内核页表中进行,反正页目录和页表的US位为0, 也
                不需要在用户空间里
73          memcpy(0xffffe000,0xfffff000,0x1000);
74          tssp->CR3 = Phyaddr(0xffffe000);
75          tssp->EFLAG= getEFLAGS();
76          tssp->EIP= *((u32*)(0x4));
77  //      *((*u32)(tssp->CR3+0x4*0x3ff)) =   prog_addr// 这一步不需要获取实际物理
        地址
78          t->next_bas=prog_addr;
79          t->state=0;
80
81
82          append_to_tcb_link(t);
83          // alloc_inst_a_page(prog_addr);         //分配一个页作页表
84          // for (int i=0; i<1024; ++i)
85          //       *((u32*)(prog_addr+i*4))= start_addr+i*4096;  // 低端12位信息待
                定
86
87  }
```

而内核的程序由于已经在 mbr.asm 中安装完毕，则只需要申请一块 TSS 的空间并填写即可。

```
1   extern u16 Load_coreself(){                         //返回TSS选择子
2           TSS_array=tss_linear_address;  //所有的TSS空间都在内核的空间进行分配。
3           struct TCB *t= &tottcb[cnttcb++];
4           t->pre=0; t->state= 0xffff;t->next_bas=0x80100000;
5           t->LDT_lim=0xffff;
```

```
        struct TSS *tssp=t->TSS_bas=TSS_array;
        t->TSS_lim= 103;
        // alloc_inst_a_page(t->TSS_bas);
        TSS_array+=0x1000;
        tssp->preTSS=0x0;tssp->CR3=getCR3();                    //填入cr3
        tssp->LDTsel=tssp->T=0;tssp->IOmap=103;

        t->TSS_sel=AddDescri_2_gdt(
                t->TSS_bas, t->TSS_lim, 0x00408900                    //制作
                    TSS的描述符，指向TSS所在空间并安装到GDT中。
        );
        append_to_tcb_link(t);
        return t->TSS_sel;
}
```

上面用到的安装 GDT，LDT，以及获取特殊寄存器的中间函数如下

```
AddDescri_2_ldt:                                ;AddDescri_2_ldt(bas, lim,
    attri, *tcb)
                                                                ; 返回选择子
                push ebx
                push ecx
                push edx
                push edi

                mov eax , [esp+0x14]
                mov ebx , [esp+0x18]
                mov ecx , [esp+0x1c]

                call flat_4gb_code_seg_sel:make_seg_descriptor
```

```
13
14              mov edi , [esp+0x20]
15                                                                  ;
                                                                        gcc
                                                                        中
                                                                        的
                                                                        str
                                                                        会
                                                                        有
                                                                        对
                                                                        齐
                                                                        的
                                                                        情
                                                                        况，
                                                                        注
                                                                        意。
16              mov ebx , [edi+0x0c]                        ;ldt base
17              xor ecx, ecx
18              mov cx , [edi+0x0a]                                 ;ldt
                    limit
19              inc cx
20              add ebx, ecx
21
22              mov [ebx], eax                                      ; bug ,
                    swap of edx:eax
23              mov [ebx+0x4], edx
24
```

```
                xor eax, eax
                mov ax, cx
                add cx, 7
                mov [edi+0xa], cx

                or ax ,0x4                              ;TI=1
                pop edi
                pop edx
                pop ecx
                pop ebx
                ret

AddDescri_2_gdt:                                ;AddDescri_2_gdt(bas,lim, attri
    )
                                                                ; 返回选择子

                push ebx
                push ecx
                push edx

                sgdt [pgdt]

                mov eax , [esp+0x10]
                mov ebx , [esp+0x14]
                mov ecx , [esp+0x18]
                call flat_4gb_code_seg_sel:make_seg_descriptor

                movzx ebx, word [pgdt]
                inc bx
```

```
                add ebx, [pgdt+2]

                mov [ebx], eax
                mov [ebx+4] , edx

                add word [pgdt] ,8

                lgdt [pgdt]
                xor eax, eax
                mov ax , [pgdt]
                shr ax , 3
                shl ax , 3

                pop edx
                pop ecx
                pop ebx
                ret
```

```
                memcpy:
                push esi
                push edi
                push eax
                push ecx

                mov edi , [esp+0x14]
                mov esi , [esp+0x18]
                mov ecx , [esp+0x1c]

        .cpying:
                mov al, [esi]                           ; mov eax [esi] is
```

```
                    error
13                  mov [edi] , al
14                  inc edi
15                  inc esi
16                  loop .cpying
17
18                  pop ecx
19                  pop eax
20                  pop edi
21                  pop esi
22                  ret
23
24  Phyaddr:
25                  push ebx
26                  mov ebx, [esp+0x8]
27                  shr ebx, 12
28                  shl ebx, 2
29                  or ebx, 0xffc00000
30                  mov eax, [ebx]
31                  and eax, 0xfffff000
32                  pop ebx
33
34                  ret
35
36  getCR3:
37                  mov eax, cr3
38                  ret
39                  global getEFLAGS
40  getEFLAGS:
```

```
41                  pushfd
42                  pop eax
43                  or eax, 0x00000200                                  ; IF
                        must be on
44                  ; and eax, 0xfffffdff                               ; IF
                        try to be off ?
45                  ret
46  Clean_partial_PDE:                                                  ;清空当前页目录
        的前半部分
47                                                                      ;
                                                                        （对
                                                                        应
                                                                        低
                                                                        2GB
                                                                        的
                                                                        局
                                                                        部
                                                                        地
                                                                        址
                                                                        空
                                                                        间）
48          pushad
49
50                  mov ebp, esp
51
52
53                  mov ebx, 0xfffff000
54                  xor esi, esi
```

```
55          .b1:
56                  mov dword [ebx+esi*4], 0x00000000
57                  inc esi
58                  cmp esi, 512
59                  jl .b1
60
61                  mov ebx, 0xfffffff8                         ;新页目录的缓存
                        也要清掉
62                  mov dword [ebx], 0x00000000
63
64                  mov eax, cr3
65                  mov cr3, eax                                ;
                        reflash TLB
66          popad
67
68          ret
```

（三）安装系统调用中断

以下是供内存调用的系统调用中断。比实验五多了一个 put_char

```
1   sys_call_handler:
2                                                               ;5
                                                                    system
                                                                    calls
                                                                    available
3                                                               ; 切换
                                                                    ds
```

```
4                   push ds
5                   push eax
6                   mov ax , flat_4gb_data_seg_sel
7                   mov ds , ax
8                   pop eax
9
10                  cmp al, 0                                       ;0  putchar ,
                        cl=char
11                  jne     .endhandle0
12                  push ecx
13                  call putchar
14                  add esp, 4
15                  jmp .endsyscall
16          .endhandle0:
17
18                  cmp al, 1                                       ;1      getchar
                         , return al=getchar()
19                  jne     .endhandle1
20                  xor eax, eax
21                  sti                                                 ; ???
                        must be open to let keyboard handle to flush the keybuf
22                  call getchar
23                  cli
24                  jmp .endsyscall
25
26          .endhandle1:
27                  cmp al, 2                                       ;2
                        simple_puts , ebx=*str  , ecx=pos<<16+col
28                  jne     .endhandle2
```

```
29                  push ecx
30                  push ebx
31                  call simple_puts
32                  add esp , 8
33                  jmp .endsyscall
34
35          .endhandle2:
36
37                  cmp al, 3                                       ;3
                        simple_putchar , ebx= color_site,  cl = ch
38                  jne     .endhandle3
39                  push ebx
40                  push ecx
41                  call simple_putchar
42                  add esp, 8
43                  jmp .endsyscall
44          .endhandle3:
45
46                  cmp al, 4                                       ;4  clock()
                        return BCD code 0x00HourMinSec
47                  jne     .endhandle4
48                  call curr_clock
49                  jmp .endsyscall
50          .endhandle4:
51
52                  cmp al, 5                                       ;5      clear
                        screen()
53                  jne     .endsyscall
54
```

```
55                call clear_screen
56
57        .endsyscall:
58                push eax
59                mov ax , [esp+4]
60                mov ds , ax
61                pop eax
62                add esp, 4
63                iretd
```

然后是往 IDT 中添加中断，包括时钟中断和系统中断，由于用户特权级为 3，那么中断门的 DPL 设为 3

```
1                 mov eax, general_exeption_handler
2                 mov bx, flat_4gb_code_seg_sel
3                 mov cx, 0x8e00                                  ;e表示中断，如
                      果用任务门则0101B，且需要提供TSS选择子。
4                 call flat_4gb_code_seg_sel:make_gate_descriptor
5
6
7                 mov ebx, idt_linear_address
8                 xor esi, esi
9
10    .idt0:
11                mov [ebx+esi*8], eax
12                mov [ebx+esi*8+4], edx
13                inc esi
14                cmp esi, 19
15                jle .idt0
16
17                mov eax, general_interrupt_handler
```

```
                mov bx, flat_4gb_code_seg_sel
                mov cx, 0x8e00
                call flat_4gb_code_seg_sel:make_gate_descriptor

                mov ebx, idt_linear_address
.idt1:
                  mov [ebx+esi*8] , eax
                  mov [ebx+esi*8+4] , edx
                  inc esi
                  cmp esi , 255
                  jle .idt1

                  ; set clock interrupt

                set_up_Idescriptor rtm_0x70_interrupt_handle, 0x70 ,0xee00
                  ; set keyboard inter
                ; set_up_Idescriptor keyboard_interrupt_handle,0x21 , 0xee00

                  ; set syscall
                set_up_Idescriptor sys_call_handler, 0x11, 0xee00       ;
                           dpl ->3
```

（四）扩展系统调用

为了能在用户程序中实现更加复杂的功能，考虑使用 C 语言.h 和 asm 头文件交叉编译共同组成头文件。这样实现的功能就能够直接供 C 用户程序使用。

```
#ifndef STDCALL
#define STDCALL

typedef unsigned char u8;
```

```
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;
#define DelayTime 200000

extern void simple_putchar(u32 , u32);
extern u32 clock();

const int dx[4]={1, 1, -1,-1}, dy[4]={1,-1,1,-1};
int curx, cury=0,curd=0;
u32 randseed;

int Delay(int delayt){int j=0;for (int i=delayt; i--;)j+=i;return j;}
u32 rand(){return randseed=randseed*16807%0xffff;}

void c_block_stone(u32 BaseX, u32 BaseY, u32 Limx,u32 Limy,u32 color){
        curx=0;cury=0;
        for (int i=0; i<10000; ++i){
                simple_putchar('*', ((curx+BaseX)*80+cury+BaseY<<16)+color);
                Delay(DelayTime);
                simple_putchar(' ', ((curx+BaseX)*80+cury+BaseY<<16)+color);
                int nx=curx+dx[curd], ny=cury+dy[curd];
                if ( nx<0 || nx>=Limx) curd^=2;
                if ( ny<0 || ny>=Limy) curd^=1;
                curx+=dx[curd]; cury+= dy[curd];
        }
}

void c_snakewind(u32 BaseX, u32 BaseY, u32 Limx, u32 Limy){
```

```
34
35          for (int step=(Limx<Limy ? Limx:Limy)>>1, i=0,cnt=0; i<step; ++i){
36          // for (int i=0; i<2; ++i){
37                  for (int j=BaseY+i;j<(int)BaseY+Limy-i; ++j, cnt=(cnt+1)%15){
38                          simple_putchar('+',((BaseX+i)*80+j<<16)+4);
39                          Delay(DelayTime/10);
40                  }
41                  for (int j=BaseX+i; j<(int)(BaseX+Limx-i); ++j,cnt=(cnt+1)%15){
42                          simple_putchar('+',(j*80+BaseY+Limy-1-i<<16)+1);
43                          Delay(DelayTime/10);
44                  }
45                  for (int j=BaseY+Limy-i-1;j>=(int)(BaseY+i); --j,cnt=(cnt+1)
                        %15){
46                          simple_putchar('+',((BaseX+Limx-1-i)*80+j<<16)+4);
47                          Delay(DelayTime/10);
48                  }
49                  for (int j=BaseX+Limx-i-1; j>=(int)(BaseX+i); --j,cnt=(cnt+1)
                        %15){   // bugs, lack of int and -1 will >=0
50                          simple_putchar('+',(j*80+BaseY+i<<16)+1);
51                          Delay(DelayTime/10);
52                  }
53          }
54          for (int i=0; i<Limx; ++i)
55                  for (int j=0; j<Limy; ++j){
56                          simple_putchar(' ', ((i+BaseX)*80+j+BaseY<<16)+0x0);
57                          Delay(DelayTime/20);
58                  }
59  }
60
```

```
#define SNAKELEN 20
const int vdx[4]={0,1,0,-1}, vdy[4]={1,0,-1,0};
void c_snakerand(u32 BaseX, u32 BaseY,  u32 Limx, u32 Limy,const u8 *info, u32 color){
    randseed=(clock()+info[0]+info[1])|1;
    u8 x[SNAKELEN+1], y[SNAKELEN+1];
    for (int i=1; i<=SNAKELEN; ++i)x[i]=(i-1)/Limy, y[i]=(i-1)%Limy;
    for (int i=SNAKELEN;i; --i)
        simple_putchar(info[i-1], ((x[i]+BaseX)*80+y[i]+BaseY<<16)+color);
    for (u8 cnt=0,d=rand()%4;;++cnt){
        if ( cnt%6==0) d=rand()%4;
        for(x[0]=x[1]+vdx[d], y[0]= y[1]+vdy[d]; x[0]>=Limx || y[0]>=Limy; x[0]=x[1]+vdx[d], y[0]= y[1]+vdy[d])
            d=(d+1)%4;
        simple_putchar(' ',((x[SNAKELEN]+BaseX)*80+BaseY+y[SNAKELEN]<<16));
        for (int i=SNAKELEN;i; --i){
            x[i]=x[i-1]; y[i]=y[i-1];
            simple_putchar(info[i-1], ((x[i]+BaseX)*80+y[i]+BaseY<<16)+color);
        }
        Delay(DelayTime/3);
    }
}
#endif
```

（五）时钟中断控制任务切换

当发生时钟中断时，需要从 TCB 链表中找出当前正在运行和未被运行的程序，对调程序运行状态之后，利用 jmp far 发起任务切换

```
extern u32 c_rtm_0x70_interrupt_handle(){
        // Out(0xa0, 0x20); Out(0x20, 0x20);
        // Out(0x70, 0x0c); In(0x71);

        buf[0] = message_cyc[curcyc++];buf[1]=0;
        curcyc&=3;
        simple_puts(buf, (24*80+79<<16) + 4);
        show_current_clock();
        if ( TCBHeader->pre ==0)return -1;
        struct TCB *curact=TCBHeader;
        while ( curact->pre) curact= curact->pre;
        curact->state=0;
        curact->pre=TCBHeader;TCBHeader=TCBHeader->pre;
        curact=curact->pre;
        curact->pre=0;curact->state=0xffFF;
        return curact;
}
```

```
                rtm_0x70_interrupt_handle:
                sti
                cli
                                        ;只是为了方便设断点
                push ds
                push eax
                mov ax , flat_4gb_data_seg_sel
                mov ds , ax
                pop eax
```

```
9
10              pushad
11              mov al,0x20                     ;中断结束命令EOI
12              out 0xa0,al                     ;向8259A从片发送
13              out 0x20,al                     ;向8259A主片发送
14
15              mov al,0x0c                     ;寄存器C的索引。且开放NMI
16              out 0x70,al
17              in al,0x71                      ;读一下RTC的寄存器C，否则只
                    发生一次中断
18                                                                          ;
                                                                            此
                                                                            处
                                                                            不
                                                                            考
                                                                            虑
                                                                            闹
                                                                            钟
                                                                            和
                                                                            周
                                                                            期
                                                                            性
                                                                            中
                                                                            断
                                                                            的
                                                                            情
                                                                            况
19              call c_rtm_0x70_interrupt_handle
```

```
20                                                                  ;
                                                                    C 语言完成任务切换，并返回下一个任务 TCB 指针
21              cmp eax, 0xffffffff
22              je .noxchg
23              jmp far  [eax+0x14]                          ; u32 TSS_bas ;
                     u16 TSS_sel
24      .noxchg:
```

```
25                  popad
26
27                  push eax
28                  mov ax , [esp+4]
29                  mov ds , ax
30                  pop eax
31                  add esp, 4
32                  iretd
```

（六）四个用户程序实现

四个用户程序结构大概类似，都是一个不同的系统调用。分别在 4 个象限显示信息。

```
1  // user0
2  #include "syscall.h"
3
4  extern void cstart(){
5          while (1)       c_block_stone(0,0, 13, 40,7);
6  }
7
8  // user 1
9  #include "syscall.h"
10
11 extern void cstart(){
12         while (1)       c_snakewind(0,40, 13, 40);
13 }
14
15
16 // user2
17 #include "syscall.h"
```

```

const u8 info[SNAKELEN+1]="OS in protect MODE——";
extern void cstart(){
        while ( 1) c_snakerand(13,0,12,40, info, 0xe);
}

//user3
#include "syscall.h"

const u8 info[SNAKELEN+1]="17341038 fuchang's***";
extern void cstart(){
        while (1)       c_snakerand(13, 40, 12,40, info, 0xa);
}
```

```
                        [bits 32]
                        program_length dd prog_end-$$
                        entry_point dd cstart

extern cstart
%include "trash_syscall.inc"
global _start
_start:
        jmp _start

                ; mov al, 4                      ; clear _screen
                ; int 0x11
                ; call user_main


;---------------
```

```
17 [section .data]
18         sign db ’this is data end’
19         prog_end:
```

五、 实验结果及分析

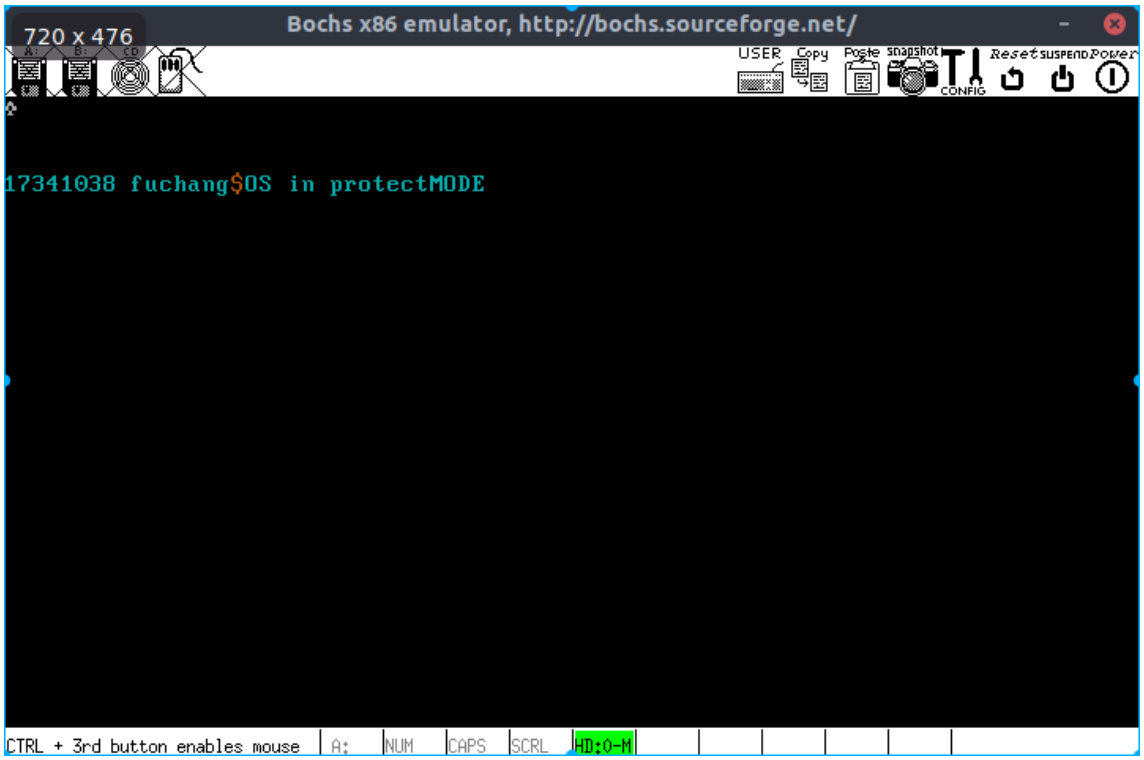

图 2: 时钟中断正式开启，在内核显示个人信息

如下图，时钟显示和风火轮照常显示。

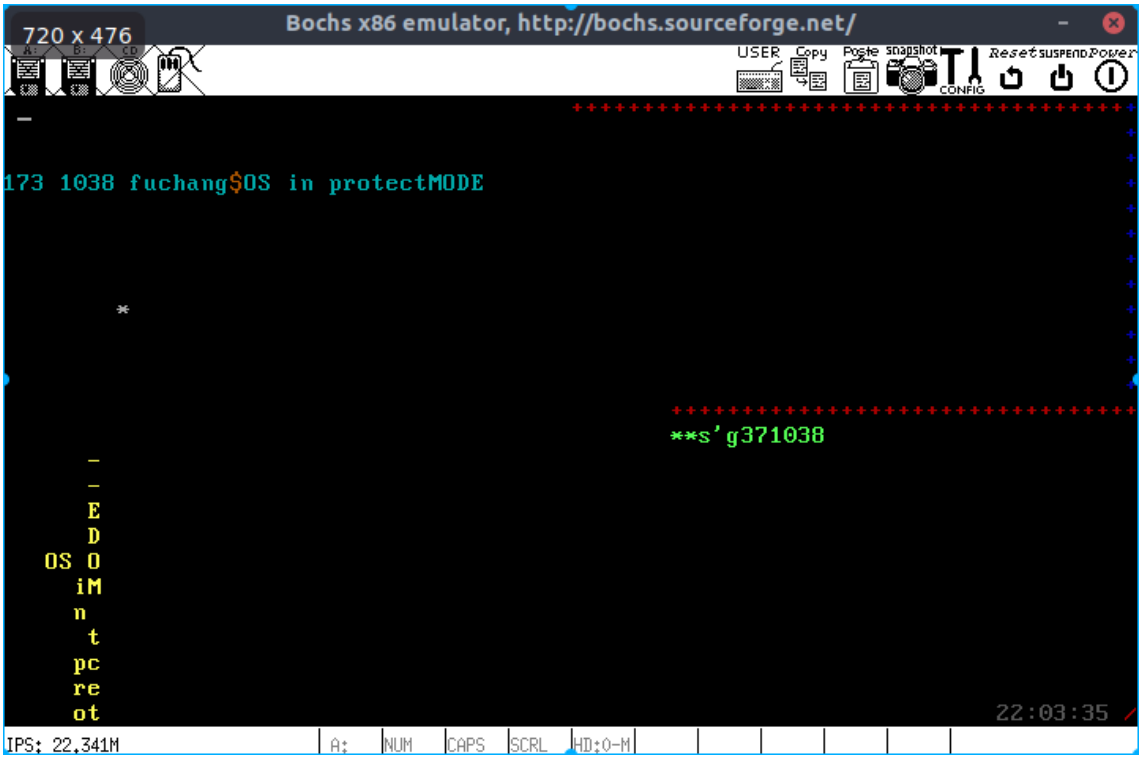

图 3: 执行过程中的截图 1，四个象限分别是弹球、环绕蛇和标识个人信息的随机游走蛇

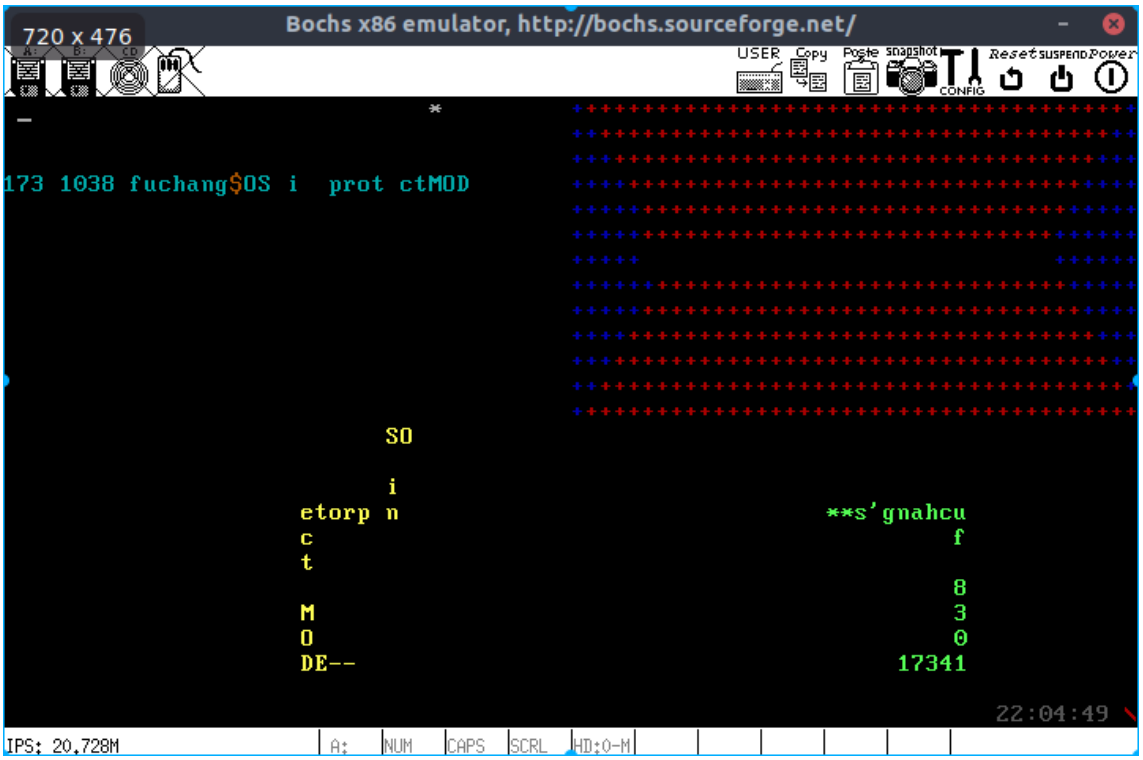

图 4: 执行过程中的截图 2，四个象限分别是弹球、环绕蛇和标识个人信息的随机游走蛇

六、 实验总结

这次实验比起前，更多的利用了 TSS、LDT 的功能。这两块的内容的设计非常繁琐，特别是 TSS，设计稍有不慎就会出事。

我在任务切换的那条语句 jmp far 中受到的阻碍最大。一条语句包含了：将通用寄存器快照保存进当前 TSS，从新 TSS 中取出新程序的选择子和页目录，然后实施跳转。整个繁琐的过程在一个语句里完成。这大大增加了我调试的难度。这也再次打击了我的侥幸心理。一条没有经过手算的程序，是不可能在机器上成功跑起来的。从抽象到代码里面隔了不小的距离

我不得不将 TCB，TSS，页表、页目录中的所有值都在 bochs 中显示出来，并和一份比较标准的代码相比较。首先就找出了 TSS 描述符填写的错误，然后再发现了自己加载用户内核程序的错误（硬盘制作的一些问题），然后就发现了自己的新分配的页需要清零否则会有很多奇怪的页被加进来....

逐步排查永远是修补工程 bug 最有力的手段，偷懒侥幸其实是一种错误的认知。

这次实验的代码量几乎可以使以前所有代码的总和了。花了一个多月的时间来实现它，其实还是非常有收获的。