中山大學

SUN YAT-SEN UNIVERSITY

操作系统原理

实验报告

实验名称	实验十二 保护模式操作系统
姓 名	潘文轩
学 号	19335163
任教老师	凌应标
助 教	梁丰洲、车鑫恺
完成日期	2021 年 7 月 16 日

实验十二　保护模式操作系统

一、实验目的

实验目的如下：

(1) 学习保护模式操作系统的特点
(2) 熟悉保护模式操作系统的实现方式
(3) 对比实模式操作系统，理解保护模式的优点

实验内容如下：

(1) 实现实模式到保护模式的跳转
(2) 构造 GDT、LDT，实现操作系统的分段内存管理
(3) 通过调用门和远返回实现段间跳转和特权级转换
(4) 构造页目录表与页表实现操作系统的分页内存管理
(5) 实现中断与陷阱
(6) 在已有的操作系统基础上引入 FAT12 文件系统组织方式
(7) 利用 Makefile 文件汇编源代码、生产软盘映像文件

实验环境：Windows10/CentOS7 + DOSBOX + VirtualBox
编译工具：NASM + GCC + LD

二、保护模式原理

在 IA32 下，CPU 有两种工作模式：实模式和保护模式。当我们打开自己的 PC，开始时 CPU 是工作在实模式下的，经过某种机制之后，才进入保护模式。在保护模式下，CPU 有着巨大的寻址能力，并为强大的 32 位操作系统提供了更好的硬件保障。

使用选择子访存

保护模式下，访问的每一段内存，都需要提前跟 CPU 声明；描述一段内存无非需要它的：

(1) 基地址
(2) 长度
(3) 扩展方式（向高地址还是向低地址）
(4) 之后会用于维护访存秩序的特权级
(5) 之后会与 Cache 有联系的“是否存在与主存”
(6) 指明默认操作数大小。
(7) 用途的粗分类

正因为上述特征，才有了描述符那样奇怪而繁杂的标志位。而描述符 (Descriptor) 和选择子 (Selector) 正是记录上述信息的载体。在保护模式中，一个程序的内存空间以段式内存方式管理，每个描述符描述了一个段的主要信息。虽然段值仍然由原来 16 位的 CS、DS 等寄存器表示，但此时它仅仅变成了一个索引，这个索引指向 GDT(Global Descriptors Table)/ LDT(Local Descriptors Table) 的一个表项，表项中详细定义了段的起始地址、界限、属性等内容。也就是说，由描述符组成的 GDT 的作用是用来提供段式存储机制，这种机制是通过段寄存器和 GDT 中的描述符共同提供的。

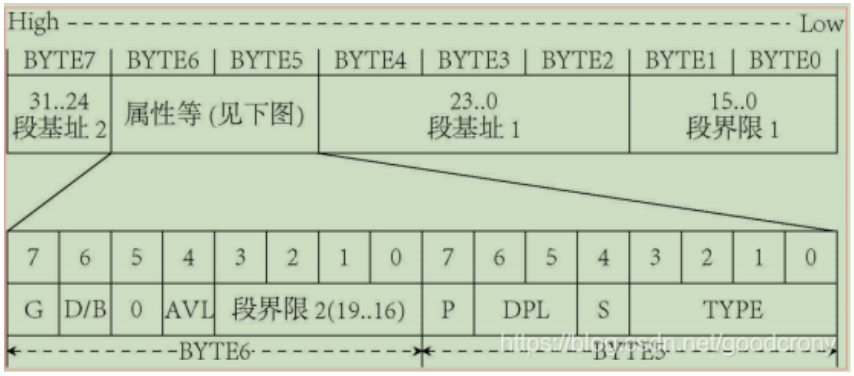

图 1: 段描述符组成

介绍完描述符和 GDT/LDT，再来介绍选择子。选择子是描述符在 GDT/LDT 中的偏移量，也是程序访问描述符的索引。选择子同样需要按表的方式组织，CPU 中有一个专门的寄存器 GDTR，指向段描述子表 GDT 的首地址；段寄存器中则只需要记录用于检索该段的选择子在表中起始位置相对 GDT 的偏移量，就足够描述一个段了。相应的，LDT 也有对应的寄存器 LDTR 来存储 LDT 的首地址。

x86 中共有 4 个控制寄存器,cr0~3。涉及到保护模式开启的另一个标志位是 cr0 的 PE 位。将该位设为 1，便进入了保护模式。

任务寄存器 TR

TR 用于寻址一个特殊的任务状态段（TaskState Segment，TSS）。TSS 中包含着当前执行任务的重要信息。

TR 寄存器用于存放当前任务 TSS 段的 16 位段选择符、32 位基地址、16 位段长度和描述符属性值。它引用 GDT 表中的一个 TSS 类型的描述符。指令 LTR 和 STR 分别用于加载和保存 TR 寄存器的段选择符部分。当使用 LTR 指令把选择符加载进任务寄存器时，TSS 描述符中的段基地址、段限长度以及描述符属性会被自动加载到任务寄存器中。当执行任务切换时，处理器会把新任务的 TSS 的段选择符和段描述符自动加载进任务寄存器 TR 中。

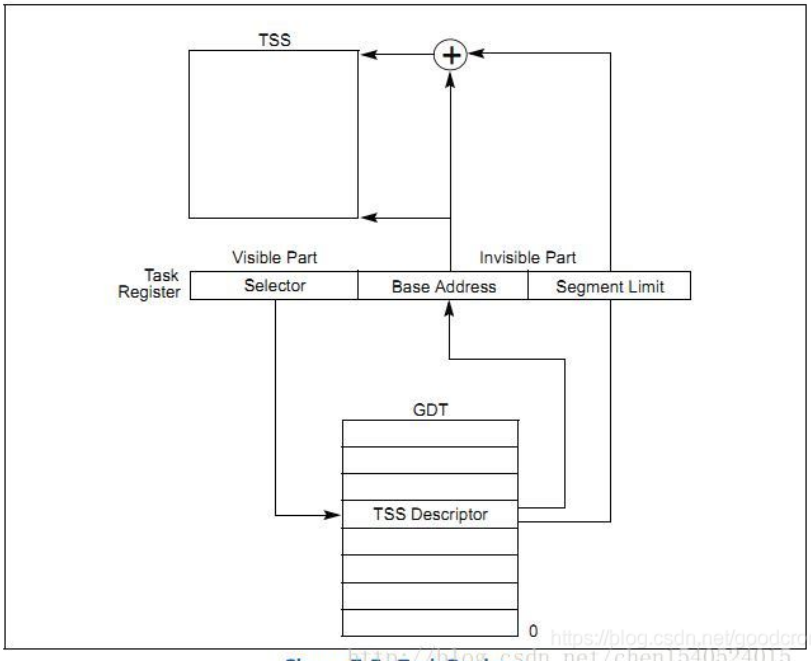

图 2: TSS 结构

分页机制

x86 提供页式访存机制。通过“页目录–页表”两层的结构，将线性 (逻辑) 地址映射到物理地址。一张页目录和一张页表和一张页都是 4kb，4B/项 *1024 项。页目录中的每一项记录页表的物理首地址，页表每一项的都记录一张 4kb 页的物理首地址。每个页目录/页表项的高 20 位表示物理地址的高 12 位 (因为页都是按 4kb 分配的)；低 12 位就用来记录和

- 高速缓存 Cache
- 页读写策略，
- 是否全局页/页表

页目录需要通过 cr3 来指向其物理首地址，并把 cr0 中的 PG 位置位，来正式开启分页机制。

中断门

门是 x86 中一类用来处理不同特权级代码之间进行控制转移的一种（保护的）格式。本质上它还是一个描述符，只是不同于代码段和数据段而已。调用门，中断门，任务门，陷阱门他们都非常相似，他们都必须包含 16 位目标代码段的选择子，32 位段内偏移量和特权级。类似于 GDT，每个中断例程都通过中断门来调用，这些 256 个中断门组成一个 1KB 的表（每项 4B），由 IDTR 来指向中断向量表的首地址以及偏移量之后每个对应中断发生时，都通过“中断向量—中断门—目标代码选择子”来实现调用中

断例程。

三、代码分析

GDT 段： GDT 段中主要存放着 GDT 表和段选择子，Descriptor 为宏定义，汇编器会根据描述符的格式和定义后面的信息组成实际的描述符。

```
;GDT段
[SECTION .gdt] ;段基址为0的段描述符需要填入实际的段基址后才能使用
;GDT                          段基址,           段界限,  描述符属性
LABEL_GDT:       Descriptor    0,                0, 0       ;空描述符
LABEL_DESC_DATA: Descriptor    0,       DataLen - 1, DA_DRW
LABEL_DESC_STACK: Descriptor   0,        TopOfStack, DA_DRWA + DA_32 ;32
    位段
LABEL_DESC_CODE32: Descriptor  0,     Code32Len - 1, DA_C + DA_32
LABEL_DESC_CODE16: Descriptor  0,           0ffffh, DA_C
LABEL_DESC_VEDIO: Descriptor 0B8000h,       0ffffh, DA_DRW + DA_DPL3
LABEL_DESC_NORMAL: Descriptor  0,           0ffffh, DA_DRW
LABEL_DESC_LDT:  Descriptor    0,        LDTLen - 1, DA_LDT
LABEL_DESC_CODE_DEST:Descriptor 0,  CodeDestLen - 1, DA_C + DA_32 +
    DA_DPL0
LABEL_DESC_RING3: Descriptor   0,  CodeRing3Len - 1, DA_C + DA_32 +
    DA_DPL3
LABEL_DESC_RING3_STACK: Descriptor 0, TopOfStackRing3, DA_DRWA + DA_32
    + DA_DPL3
LABEL_DESC_TSS:  Descriptor    0,        TSSLen - 1, DA_386TSS
LABEL_DESC_PAGE_DIR:Descriptor PageDirBase, 0fffh, DA_DRW
LABEL_DESC_PAGE_TBL:Descriptor PageTblBase, 8000h, DA_DRW

;门                           目标选择子, 偏移, PCount, 属性
LABEL_CALL_GATE_TEST: Gate  SelectorDest, 0,     0, DA_386CGate +
    DA_DPL3

GdtLen equ $ - LABEL_GDT
GdtPtr dw  GdtLen - 1    ;段界限
       dd  0             ;段基址（待填入）



;GDT选择子
```

```
SelectorData  equ LABEL_DESC_DATA  - LABEL_GDT
SelectorStack equ LABEL_DESC_STACK - LABEL_GDT
SelectorCode32 equ LABEL_DESC_CODE32 - LABEL_GDT
SelectorCode16 equ LABEL_DESC_CODE16 - LABEL_GDT
SelectorVedio equ LABEL_DESC_VEDIO - LABEL_GDT
SelectorNormal equ LABEL_DESC_NORMAL - LABEL_GDT
SelectorLDT   equ LABEL_DESC_LDT   - LABEL_GDT
SelectorDest  equ LABEL_DESC_CODE_DEST - LABEL_GDT
SelectorRing3 equ LABEL_DESC_RING3 - LABEL_GDT + SA_RPL3
SelectorStackRing3 equ LABEL_DESC_RING3_STACK - LABEL_GDT + SA_RPL3
SelectorTSS   equ LABEL_DESC_TSS   - LABEL_GDT
SelectorPageDir equ LABEL_DESC_PAGE_DIR - LABEL_GDT
SelectorPageTbl equ LABEL_DESC_PAGE_TBL - LABEL_GDT

;门选择子
SelectorCallGateTest equ LABEL_CALL_GATE_TEST - LABEL_GDT + SA_RPL3
;end of [SECTION .gdt]
```

中断门和陷阱门：

```
;IDT
[SECTION .idt]
ALIGN 32
[BITS 32]
LABEL_IDT:
;      门                目标选择子,      偏移,    DCount, 属性
%rep 32
      Gate           SelectorCode32, SpuriousHandler, 0, DA_386IGate
%endrep
.020h: Gate          SelectorCode32, ClockHandler,  0, DA_386IGate
%rep 95
      Gate           SelectorCode32, SpuriousHandler, 0, DA_386IGate
%endrep
.080h: Gate          SelectorCode32, UserIntHandler, 0, DA_386IGate

IdtLen     equ $ - LABEL_IDT
IdtPtr     dw  IdtLen - 1 ;段界限
           dd  0          ;基地址
;end of [SECTION .idt]
```

进入保护模式：

```
    ...
    初始化段描述符
    ...
;加载GDTR
    xor eax, eax
    mov ax, ds
    shl eax, 4
    add eax, LABEL_GDT
    mov dword [GdtPtr + 2], eax
    lgdt [GdtPtr]

    ;打开A20地址线
    ;cli            ;在保护模式下一直关中断，中断以其他形式实现
    in al, 92h
    or al, 00000010b
    out 92h, al

    ;cr0置PE位
    mov eax, cr0
    or eax, 1
    mov cr0, eax

    ;进入保护模式
    jmp dword SelectorCode32:0 ;本指令将SelectorCode32装载进入cs，dword不
        能少
```

退出保护模式：

```
    mov eax, cr0         ;修改cr0寄存器
    and eax, 07ffffffeh  ;PE = 0, PG = 0
    mov cr0, eax         ;表示退出保护模式和分页机制
    ...
LABEL_REAL_ENTRY: ;保护模式跳转为实模式
    ;实模式下恢复段寄存器
    mov ax, cs
    mov ds, ax
```

```
    mov ss, ax
    mov es, ax
    mov gs, ax

    mov sp, [RealModeSP] ;将栈恢复为实模式的栈

    ;IDTR相关设置
    lidt [_SavedIDTR]    ;恢复IDTR原值
    mov al, [_SavedIMREG] ;恢复原中断屏蔽寄存器(IMREG)原值
    out 21h, al

    ;关闭A20地址线
    in al, 92h
    and al, 11111101b
    out 92h, al

    sti             ;恢复实模式中断
```

启动分页机制：

本次实验实现的是最简单的分页机制，即线性地址映射前后相同。其作用在于展示分页机制如何实现，同时其他非相等映射实现的基础也是相等映射。对本分页机制的改进方向还有很多，例如增加调页算法和缺页中断，这些都是完整的分页机制所必须实现的。本分页机制的大致流程为：

(1) 根据当前内存计算页框数 (内存在之前已通过 int 15h 内存信息中断获得)
(2) 根据页框数 (页表表项) 计算页目录表的表项数
(3) 按照一定的属性写入页目录表的表项 (PDE)，指向的页表地址使页表相连 (起始地址相差 4096 字节)
(4) 按照一定的属性写入页表表项 (PTE)，相邻的页在物理空间上相邻
(5) 将页目录表的基址载入 cr3，同时将 cr0 的 PG 位置位，表示开启分页机制

```
;启动分页机制------------------------------------------------------------
;简单起见，所有的线性地址等于对应的物理地址，即分页机制开启前后物理地址不变
SetupPaging:            ;计算当前内存并计算页表大小
    xor edx, edx
    mov eax, [dwMemSize]
    mov ebx, 400000h
    div ebx
```

```
    mov ecx, eax      ;ecx存放根页表数
    test edx, edx
    jz .no_remainder ;能整除
    inc ecx           ;如果不能整除, 则需要增加一个页表
.no_remainder:
    push ecx          ;暂时保存页表数量

    ;初始化页目录表
    mov ax, SelectorPageDir
    mov es, ax
    xor edi, edi
    xor eax, eax
    mov eax, PageTblBase + PG_P + PG_USU + PG_RWW
.1:
    stosd
    add eax, 4096
    loop .1

    ;初始化页表
    mov ax, SelectorPageTbl
    mov es, ax
    pop eax           ;取出页表数
    mov ebx, 1024
    mul ebx           ;页表项PTE数 = 1024 * 页表数
    mov ecx, eax      ;循环次数 = PTE数
    xor edi, edi
    xor eax, eax
    mov eax, PG_P + PG_USU + PG_RWW
.2:
    stosd
    add eax, 4096
    loop .2

    mov eax, PageDirBase
    mov cr3, eax      ;将页目录表基址载入cr3
    mov eax, cr0
    or eax, 080000000h ;PG = 1
    mov cr0, eax
```

```
    jmp short .3
.3:
    nop
    ret
;分页机制启动完成----------------------------------------------------------
```

中断处理程序：在与中断有关的函数中，Init8259A 函数用于初始化两块 8259A 芯片，使得它们能够正确响应程序需要的中断。SetRealMode8259A 则是将两块 8259A 芯片的状态恢复为实模式下的状态。后面的 ClockHandler、UserIntHandler、SpuriousHandler 三个函数分别为时钟中断，系统中断和其他中断的处理程序，在此只有比较简易的实现，但是可以很容易利用这些接口扩展其功能。

```
;Init8259A-----------------------------------------------------------------
Init8259A:
    mov al, 011h
    out 020h, al      ;主8259, ICW1
    call io_delay

    out 0a0h, al      ;从8259, ICW1
    call io_delay

    mov al, 020h      ;IRQ0对应中断向量0x20
    out 021h, al      ;主8259, ICW2
    call io_delay

    mov al, 028h      ;IRQ8对应中断向量0x28
    out 0a1h, al      ;从8259, ICW2
    call io_delay

    mov al, 04h       ;IR2对应从8259
    out 021h, al      ;主8259, ICW3
    call io_delay

    mov al, 02h       ;对应主8259的IR2
    out 0a1h, al      ;从8259, ICW3
    call io_delay

    mov al, 01h
```

```
    out 021h, al      ;主8259, ICW4
    call io_delay

    out 0a1h, al      ;从8259, ICW4
    call io_delay

    ;mov al, 11111111b ;屏蔽主8259所有中断
    mov al, 11111110b ;仅开启时钟中断
    out 021h, al      ;主8259, OCW1
    call io_delay

    mov al, 11111111b ;屏蔽从8259所有中断
    out 0a1h, al      ;从8259, OCW1
    call io_delay
    ret
;Init8259A-----------------------------------------------------------

;SetRealMode8259A----------------------------------------------------
SetRealMode8259A:
    mov ax, SelectorData
    mov fs, ax

    mov al, 017h
    out 020h, al      ;主8259, ICW1
    call io_delay

    mov al, 08h       ;IRQ0对应中断向量0x8
    out 021h, al      ;主8259, ICW2
    call io_delay

    mov al, 01h
    out 021h, al      ;主8259, ICW4
    call io_delay

    mov al, [fs:SavedIMREG] ;恢复原中断屏蔽寄存器(IMREG)的原值
    out 021h, al
    call io_delay
    ret
```

```
;SetRealMode8259A------------------------------------------------------

io_delay:
    nop
    nop
    nop
    nop
    ret

;int handler-----------------------------------------------------------
_ClockHandler:
ClockHandler  equ _ClockHandler - $$
    mov ax, SelectorData
    mov ds, ax
    mov bh, 0fh
    mov dx, word [wheel_cnt]
    ;mov dx, 16
    cmp dx, 0
    jg cmp_wheel_cnt
    mov word [wheel_cnt], 16
    mov dx, 16
cmp_wheel_cnt:              ;控制速度：每中断16次，风火轮旋转一周
    cmp dx, 16
    je wheel1
    cmp dx, 12
    je wheel2
    cmp dx, 8
    je wheel3
    cmp dx, 4
    je wheel4
    jmp exit_timer          ;若当前计数器不是4的倍数，不更新风火轮
wheel1:
    mov bl, '|'
    jmp print_wheel
wheel2:
    mov bl, '/'
    jmp print_wheel
wheel3:
```

```
    mov bl, '-'
    jmp print_wheel
wheel4:
    mov bl, '\'
    jmp print_wheel
print_wheel:                ;输出当前风火轮的状态
    mov word [gs:(80 * 24 + 79) * 2], bx
exit_timer:
    dec word [wheel_cnt]
    mov al, 20h
    out 020h, al

    iretd




_UserIntHandler:
UserIntHandler equ _UserIntHandler - $$
    mov ah, 0Ch
    mov al, 'U'
    mov [gs:((80 * 0 + 78) * 2)], ax ;在屏幕第0行第78列打印字符'U'
    iretd

_SpuriousHandler:
SpuriousHandler equ _SpuriousHandler - $$
    mov ah, 0Ch
    mov al, 'S'
    mov [gs:((80 * 0 + 79) * 2)], ax ;在屏幕第0行第79列打印字符'S'
    iretd
```

Makefile：

本次实验的 Makefile 文件如下。主要思想是使用 Linux 自带的回环设备挂载功能，实现 FAT12 文件系统表项的填充：

```
BIN = boot.bin
COM = kernel.com
IMG = protected_mode.img
FLOPPY = /mnt/floppy
```

```
.PHONY: clean clear all

all: $(IMG) clear

$(IMG): $(BIN) $(COM)
   rm -f *.img
   mkfs.msdos -C $@ 1440
   dd if=boot.bin of=$@ conv=notrunc
   #dd if=kernel.com of=$@ seek=1 conv=notrunc
   sudo mount -o loop $(IMG) $(FLOPPY)
   cp $(COM) $(FLOPPY)
   sudo umount $(FLOPPY)

%.bin: %.asm
   nasm -fbin -o $@ $<

%.com: %.asm
   nasm -fbin -o $@ $<

clear:
   rm -f $(BIN) $(COM)

clean:
   rm -f $(BIN) $(COM) $(IMG)
```

四、实验结果

如图3所示，当前的保护模式 LumosOS 实现的功能十分简单，几乎只能展示保护模式各种结构能够正确运行的结果。结果分析如下：

(1) 正上方的红色'3'：表示能够正确通过压栈和远跳转 retf 指令实现从特权级为 0 的代码段跳转至特权级为 3 的代码段 Ring3

(2) 正上方的红色'C'：表示能够正确通过调用门 (call gate) 实现从特权级为 3 的代码段 Ring3 跳转至特权级为 0 的代码段

(3) 右上角的红色'U' 和'S'：分别表示系统中断 int 80h 和其他未定义中断能够正确响应并被处理

(4) 右下角的'/'：表示时钟中断正确运行。在连续的时间段中，风火轮持续转动

(5) 中间的内存信息：表示当前保护模式的内存使用情况分析，给出了当前的基址和长度以及使用状态 (1 表示可用，2 表示不可用)

此外，还有 LDT 局部任务的输出未展示出来。上述的输出也大概概括了本次实验实现的功能。对于因时间关系未能实现的功能，我将会在暑假逐一完善。

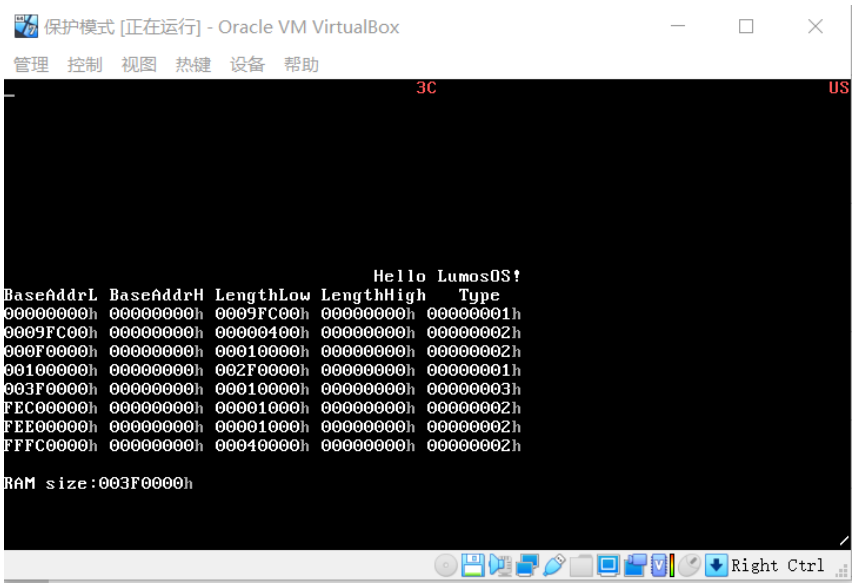

图 3: 保护模式 LumosOS 展示

五、实验感想

保护模式 LumosOS 的实现是这么多个实验以来耗费了我最多时间的一次实验，也证明老师说的“保护模式的实现是一座大山，门槛很高”的话的正确性。但是，当我真正从自第一个实验使用至今的实模式进入保护模式时，内心真是十分激动。当我用着与实模式截然不同的访存方式和数据结构在显存中输入第一个字符时，我便明白，我对于操作系统的探索和实践只是刚刚开始，并不会随着一个学期操作系统课程结束而结束。

为什么在实模式下做了那么多个实验后还要选择进入保护模式？这是身边同学问我最多的一个问题，也是我需要想明白的问题。首先，没有在一开始进入保护模式很大程度上因为我在操作系统课程的前半阶段对操作系统没有一个清晰的概念，不明白保护模式与实模式之间的区别是什么。当我意识到两者从实验三便出现截然不同的效果时，我们的实验进度已经到达实验五了。当时我便暗暗计划着，从实验七扩展实验不限 ddl 时，便从头开始完成保护模式的操作系统。

前面只回答了为什么那么晚开始，还没解释为什么要开始。其实开始的契机，是计网老师在课堂上无意间提到的一生一芯项目。老师介绍的“自己 CPU-> 在自己的 CPU 上运行自己的操作系统-> 在 CPU 上设计自己的编程语言-> 在自己的平台上运行自己的程序”这样一个流程深深吸引了我，也坚定了我在这条道路上尝试的信念。将这几个

步骤拆解开，实际上和每门课程的课程设计类似，只不过在设计难度上会比普通的作业高上不少，但也不是不能完成。所以我想试试我能在这条路上走多远，而这样的基础，便是在每一门课程的课程作业中都付出更多的时间和精力，达到老师实验要求以上的成果。

上面说了那么多与本次实验无关的话，再回来说说这次实验遇到的问题。刚刚进入保护模式后，第一个感觉是，调试相比于实模式更加困难了：由于保护模式无法在运行前得到物理地址，因此无法提前设置断点进行调试，为调试增加了难度。同时，在6的代码处，经过单步调试得到的结果图5中，可以看到在程序运行时出现了不知名代码，干扰了程序的正常执行，导致操作系统崩溃。经过肉眼一遍遍观察，发现原来是在 GDT 中 32 位代码段忘记加 DA_32 属性，导致跳转时将 32 位代码段当作 16 位代码段跳入，出现错误。此外，32 位栈段也需要加上该属性，否则也会出现不知名错误。

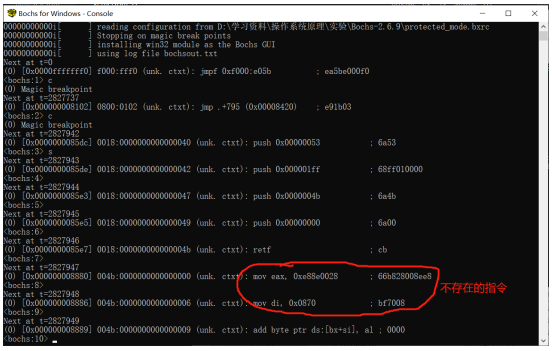

图 5: 反汇编出现莫名代码

```
22  LABEL_DESC_CODE_DEST:Descriptor     0,    CodeDestLen - 1, DA_C + DA_DPL0
23  LABEL_DESC_RING3:   Descriptor      0,  CodeRing3Len - 1, DA_C + DA_DPL3
24  LABEL_DESC_RING3_STACK: Descriptor  0,   TopOfStackRing3, DA_DRWA + DA_DPL3
```

图 6: 莫名代码错误出现的代码段

在《Orange'S》中，常常会用到 stosd/lodsd 指令进行内存赋值。但是，在使用上述两条指令时一定要注意设置 ES 的值！例如 stosd: eax -> dword ptr [es:edi]; 中指令完成后 edi += 4，lodsd: dword ptr [es:esi] -> eax; 指令完成后 esi += 4。这些都是指令的一些细节且是编程者在使用时必须知道的知识。如果出错，就会出现如图8所示的内存写入错误。

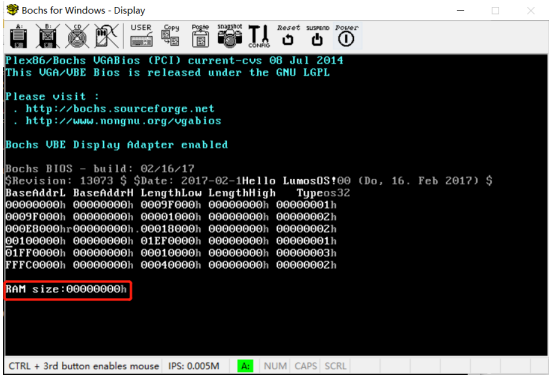

图 7: 获取内存信息失败

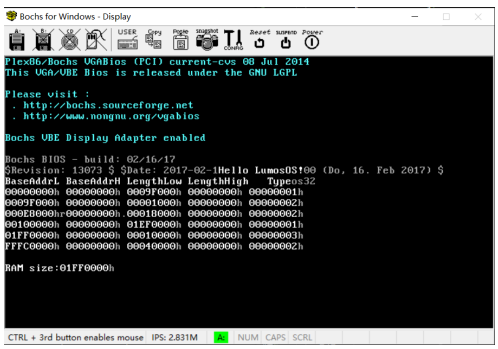

图 8: 获取内存信息成功

在中断和中断处理程序中，我也遇到了一些问题。在设置好 IDT 和中断处理程序后，使用 int 20h 出现如图9所示的系统崩溃，而不使用 int 20h 则不显示任何信息。经过排查，发现是 ICW1 设置时设置错误，误将时钟中断屏蔽，导致时钟中断无法自动触发；同时在单步调试中，变量 wheel_cnt 无法访问 (由于使用了实模式的绝对地址而非段内相对偏移地址)。两个问题解决后，时钟中断正常运行。

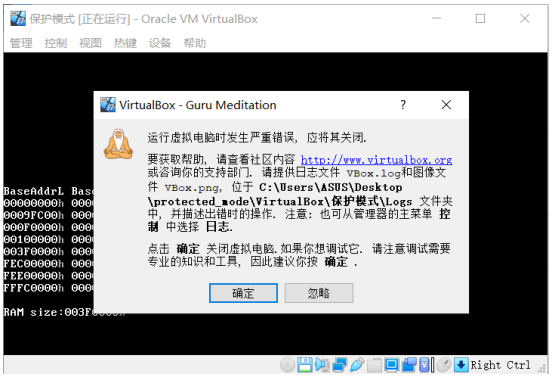

图 9: 使用 int 20h 指令系统崩溃

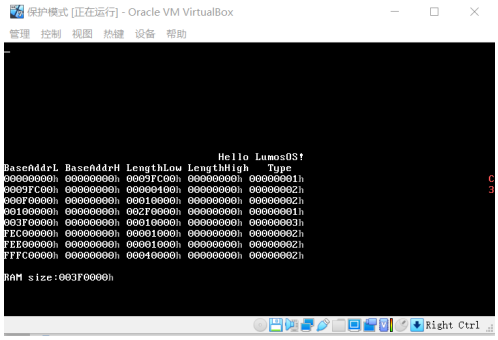

图 10: 不使用 int 20h 指令无显示

```
442     ;mov al, 11111111b   ;屏蔽主8259所有中断
443     mov al, 11111101b    ;仅开启时钟中断
444     out 021h, al         ;主8259, OCW1
445     call io_delay
```

图 11: ICW1 错误

```
494   ;int handler--------------------------------
495   _ClockHandler:
496   ClockHandler    equ _ClockHandler - $$
497       mov ax, SelectorData
498       mov ds, ax
499       mov bh, 0fh
500       mov dx, word [wheel_cnt]
501       ;mov dx, 16
502       cmp dx, 0
503       jg cmp_wheel_cnt
504       mov word [wheel_cnt], 16
505       mov dx, 16
```

图 12: 无法访问变量 wheel_cnt

一个学期的操作系统实验已接近尾声，本次实验应该也是我提交的最后一次实验。有一点遗憾的是，由于最后几个星期在同时进行扩展实验和保护模式的实现，导致没办法将实模式中的许多功能转移到保护模式中，有些功能例如加载 ELF 文件、键盘中断等也只是在《Orange'S》中学习过但没来得及实现。但是保护模式做到这里，老师上课时讲到的很多东西我已经有了新的自己的认识了，而且我认为保护模式对理解段页式存储管理非常重要。当我真正实现了一次保护模式的段页式管理机制后，发现段页式管理机制不过如此。此外还有调度算法可以加入到操作系统中，使我们的操作系统更加接近一个现实可用，有实际意义的操作系统。

趁考试完后的一个星期空闲时间，我也抽空看了一下一个学期以来我做过的操作系统实验：从最基础的 x86 汇编小程序弹弹球，到独立内核实现（这个我认为是除保护模式外最大的一个坎），再到后来的中断、进程操作和文件系统，我发现一个操作系统的实现并没有开学时想的那么难。只要一步一个脚印跟着老师的进度，即使不进入保护模

式，我也能收获一个比较完整的操作系统，学到很多理论课没办法理解的知识，或许这就是实验课的魅力吧。做到现在，虽然我知道我目前的进度离优秀差距还很远，但我可以很自豪地说一句：我实现过一个简单的操作系统，关于操作系统的基本问题我都略有耳闻，相比起一个学期前的我我在操作系统这个知识领域有了很大的进步。

最后也谢谢老师和助教的辛勤付出，老师精心设计的实验步骤和课堂上细致的讲解，为我理解、实现一个操作系统起到了很大的帮助。也希望老师和助教能在接下来的生活中一帆风顺、心想事成。

参考资料

[1] https://blog.csdn.net/goodcrony/article/details/88122934 一个操作系统的实现
[2]《Orange'S：一个操作系统的实现》于渊
[3] https://zhuanlan.zhihu.com/p/121807427 FAT12 文件系统介绍