

- 1. 实验要求
- 2. 预备知识与实验环境
 - 读写I/O端口
 - 读写硬盘
- 3. 实验任务
- 4. 实验步骤/关键代码/实验结果
 - 实验任务一
 - 实验任务二
 - 实验任务三
 - 初始状态
 - 1.准备GDT，用lgdt指令加载GDTR信息。
 - 2.打开第21根地址线。
 - 3.开启cr0的保护模式标志位。
 - 4.远跳转，进入保护模式。
 - 5.保护模式状态
 - 实验任务四
 - 1.清屏(实模式)
 - 2.进入保护模式(具体步骤看实验任务三)
 - 3.加载段选择子
 - 4.输出string
- 5. 总结(对实验过程中遇到的问题进行总结，可以提出对实验设置的改进意见)
- 6. 参考资料清单



实验课程: 操作系统

实验名称: 从实模式到保护模式

专业名称: 计算机科学与技术

学生姓名: 吴臻

学生学号: 21307371

实验地点: 实验中心大楼D栋501

实验成绩:

报告时间: 2023/3/26

1. 实验要求

学习到如何从16位的实模式跳转到32位的保护模式，然后在平坦模式下运行32位程序。同时将学习到如何使用I/O端口和硬件交互

2. 预备知识与实验环境

读写I/O端口

每一个端口在I/O电路中都会被统一编址。例如，主硬盘分配的端口地址是0x1f00x1f7，从硬盘分配的端口地址是0x1700x177，因为端口是独立编址的，因此我们无法使用mov指令来对端口赋值，我们使用的是in,out指令。读端口使用in指令，写端口使用out指令

```
; in指令
in al, 0x21 ; 表示从0x21端口读取一字节数据到al
in ax, 0x21 ; 表示从端口地址0x21读取1字节数据到al，从端口地址0x22读取1字节到ah

mov dx,0x379
in al, dx ; 从端口0x379读取1字节到al

; out指令
out 0x21, al ; 将al的值写入0x21端口
out 0x21, ax ; 将ax的值写入端口地址0x21开始的连续两个字节
mov dx, 0x378
out dx, ax ; 将ah和al分别写入端口0x379和0x378
```

in指令的源操作数只能是立即数或dx，目的操作数只能是ax和al；out指令的源操作数只能是al或ax，目的操作数只能是立即数或dx。

读写硬盘

1. 设置起始的逻辑扇区号 逻辑扇区号是被分成4段写入端口的。其中，逻辑扇区的07位被写入0x1F3端口，815位被写入0x1F4端口，16~23位被写入0x1F5端口，最后4位被写入0x1F6端口的低4位

2. 将要读取的扇区数量写入**0x1F2**端口
3. 向**0x1F7**端口写入**0x20**，请求硬盘读。
4. 等待其他读写操作完成
5. 若在第4步中检测到其他操作已经完成，那么我们就可以正式从硬盘中读取数据

3. 实验任务

1. 加载bootloader，总结bootloader的作用是什么
2. 将LBA28读取硬盘的方式换成CHS读取，同时给出逻辑扇区号向CHS的转换公式
3. 进入保护模式”一节，使用gdb或其他debug工具在进入保护模式的4个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这4个步骤
4. 在进入保护模式后，按照如下要求，编写并执行一个自己定义的32位汇编程序，要求简单说一说你的实现思路，并提供结果截图。使用两种不同的自定义颜色和一个自定义的起始位置(x,y)，使得bootloader加载后，在显示屏坐标(x,y)处开始输出自己的学号+姓名拼音首字母缩写，要求相邻字符前景色和背景色必须是相互对调的。公告图片中提供了学号为21307233，姓名为宋小宝，自定义位置(12,12)的输出样式，仅供参考

4. 实验步骤/关键代码/实验结果

实验任务一

要求：加载bootloader，总结bootloader的作用是什么

步骤：根据预备知识中的读写I/O端口和读写硬盘，将参数读入0x1f0~0x1f7，然后将硬盘内容放入ds:bx

```
mov dx, 0x1f3
out dx, al    ; LBA地址7~0

inc dx        ; 0x1f4
mov al, ah
out dx, al    ; LBA地址15~8

mov ax, cx

inc dx        ; 0x1f5
out dx, al    ; LBA地址23~16

inc dx        ; 0x1f6
```

```

mov al, ah
and al, 0x0f
or al, 0xe0    ; LBA地址27~24
out dx, al

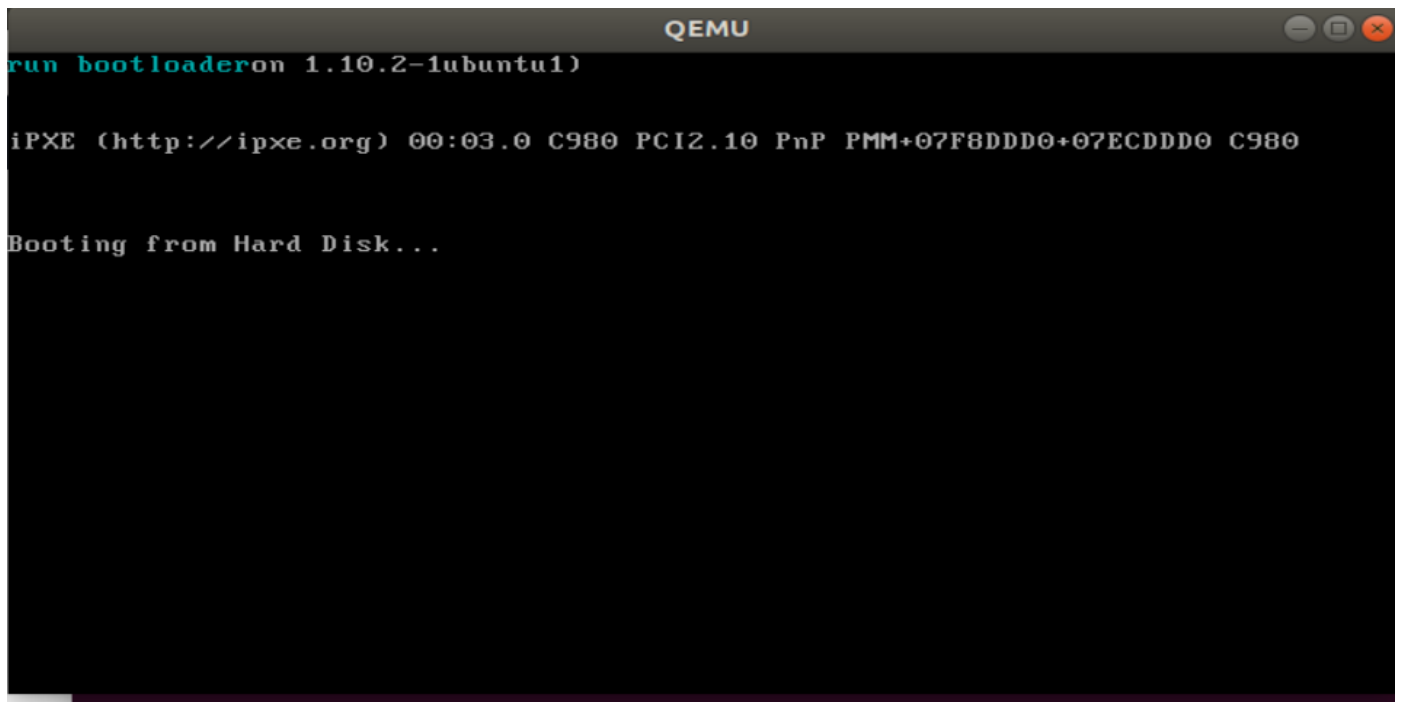
mov dx, 0x1f2
mov al, 1
out dx, al    ; 读取1个扇区

mov dx, 0x1f7    ; 0x1f7
mov al, 0x20    ; 读命令
out dx, al

; 等待处理其他操作
.waits:
in al, dx        ; dx = 0x1f7
and al, 0x88
cmp al, 0x08
jnz .waits

; 读取512字节到地址ds:bx
mov cx, 256    ; 每次读取一个字, 2个字节, 因此读取256次即可
mov dx, 0x1f0
.readw:
in ax, dx
mov [bx], ax
add bx, 2
loop .readw

```



MBR加载bootloader bootloader的作用：输出“run bootloader”

实验任务二

要求：将**LBA28**读取硬盘的方式换成**CHS**读取，同时给出逻辑扇区号向**CHS**的转换公式

步骤:将**LBA**转换为**CHS**,然后将相关参数放入寄存器,然后调用**INT 13h**中断

柱面、磁头从0开始，扇区从1开始,LBA编号从0开始

以C、H、S分别表示当前硬盘的柱面号、磁头号、扇区号，CS、HS、SS分别表示起始柱面号、磁头号、扇区号，PS表示每磁道扇区数，PH表示每柱面总的磁道数。一般情况下，CS=0，HS=0，SS=1，PS=63，PH=18 $C = LBA \text{ DIV } (PH \times PS) + CS$ $H = (LBA \text{ DIV } PS) \text{ MOD } PH + HS$ $S = LBA \text{ MOD } PS + SS$

```
    push ax                ;保证逻辑扇区号不会被改变
    push cx
;LBA -> CHS
    xor dx, dx
    div word[SectorsPerTrack] ;得到 (LBA DIV PS) 和 (LBA MOD PS)
    inc dl
    mov byte[sector],dl      ;扇区编号从1开始
    xor dx, dx
    div word [HeadsPerCylinder] ;得到 (LBA DIV PS) MOD PH 和 LBA DIV
(PH*PS)
    mov byte [head], dl
    mov byte [cylinder], al

;
    mov ah, 02h            ; AH = 02h, 读取扇区
    mov al, 01h            ; AL = 01h, 读取一个扇区
    mov ch, [cylinder] ; cylinder是逻辑扇区号转换后的柱面号
    mov dh, [head]       ; head是逻辑扇区号转换后的磁头号
    mov cl, [sector]     ; sector是逻辑扇区号转换后的扇区号
    mov dl, 80h          ; DL = 80h, 读取80h号硬盘
    int 0x13             ; 调用INT 13h中断读取硬盘数据
    add bx,512
    pop cx
    pop ax
    ret

cylinder db 0
head db 0
sector db 0
SectorsPerTrack dw 63
HeadsPerCylinder dw 18
```

实验任务三

要求：进入保护模式”一节，使用gdb或其他debug工具在进入保护模式的4个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这4个步骤

初始状态

终端

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
bootloader.asm
13      inc esi
14      add ebx,2
15      loop output_bootloader_tag
16
17      ;空描述符
B+> 18      mov dword [GDT_START_ADDRESS+0x00],0x00
19      mov dword [GDT_START_ADDRESS+0x04],0x00
20
21      ;创建描述符，这是一个数据段，对应0~4GB的线性地
22      mov dword [GDT_START_ADDRESS+0x08],0x0000ffff      ; 基地址为0，
23      mov dword [GDT_START_ADDRESS+0x0c],0x00cf9200      ; 粒度为4KB，
24
25      ;建立保护模式下的堆栈段描述符
```

remote Thread 1 In: output bootloader_tag L18 PC: 0x7e24

esi 0x7eec 32492

---Type <return> to continue, or q <return> to quit---

Quit

(gdb) x/5xg 0x8800

0x8800: 0x0000000000000000 0x0000000000000000

0x8810: 0x0000000000000000 0x0000000000000000

0x8820: 0x0000000000000000

(gdb)

1.准备GDT，用lgdt指令加载GDTR信息。

可以从下图看到GDT已加载到相应地址

```
终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

bootloader.asm
36
37 ;初始化描述符表寄存器GDTR
38 mov word [pgdt], 39 ;描述符表的界限
39 lgdt [pgdt]
B+ 40
> 41 in al,0x92 ;南桥芯片内的端口
42 or al,0000_0010B
43 out 0x92,al ;打开A20
44
45 cli ;中断机制尚未工作
46 mov eax,cr0
47 or eax,1
48 mov cr0,eax ;设置PE位

remote Thread 1 In: output_bootloader_tag L41 PC: 0x7e89
Continuing.

Breakpoint 3, output_bootloader_tag () at bootloader.asm:41
(gdb) x/5xg 0x8800
0x8800: 0x0000000000000000 0x00cf92000000ffff
0x8810: 0x0040960000000000 0x0040920b80007fff
0x8820: 0x00cf98000000ffff
(gdb) █
```

2.打开第21根地址线。

在实模式下，A20Gate是关闭的，意味着只能使用20根地址线，需要通过打开A20Gate,访问第21根以上的总线 打开A20Gate，可以使用到32位的地址总线，内存地址访问也达到了 $1 \ll 32$ 的4G范围。

```
终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

Register group: general
eax      0x302    770      ecx      0x0      0
edx      0x80     128      ebx      0x1c     28
esp      0x7c00   0x7c00   ebp      0x0     0x0
esi      0x7eec   32492    edi      0x0     0
ebp      0x7e8f   0x7e8f   eflags   0x202    [ IF ]
cs       0x0      0        ss       0x0     0
ds       0x0      0        es       0x0     0
fs       0x0      0        gs       0xb800  47104

bootloader.asm
33 ;创建保护模式下平坦模式代码段描述符
34 mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基地址为0，段界限为0xFFFFF
35 mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb，代码段描述符
36
37 ;初始化描述符表寄存器GDTR
38 mov word [pgdt], 39 ;描述符表的界限
39 lgdt [pgdt]
B+ 40
41 in al,0x92 ;南桥芯片内的端口
42 or al,0000_0010B
43 out 0x92,al ;打开A20
B+ 44
> 45 cli ;中断机制尚未工作

remote Thread 1 In: output_bootloader_tag
(gdb) c
Continuing.

Breakpoint 4, output_bootloader_tag () at bootloader.asm:45
(gdb) layout regs
(gdb) fs src
Focus set to src window.
(gdb) █
```

3.开启cr0的保护模式标志位。

CR0寄存器是一个32位的寄存器

设置CR0寄存器的最高位为0，最低位为1，则可以进入保护模式。

CR0寄存器的作用: 改变段寻址方式，使用段描述符方式寻址。实模式指令的操作数默认为16位，保护模式指令的操作数默认为32位。

```
终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Register group: general
eax      0x11      17      ecx
edx      0x80      128     ebx
esp      0x7c00    0x7c00   ebp
esi      0x7eec    32492   edi
eip      0x7e9a    0x7e9a <output_bootloader_tag+132> eflags
cs       0x0       0      ss
ds       0x0       0      es
fs       0x0       0      gs

42      or al,0000_0010B
43      out 0x92,al      ;打开A20
44
45      cli              ;中断机制尚未工作
46      mov eax,cr0
47      or eax,1
48      mov cr0,eax      ;设置PE位
49
50      ;以下进入保护模式
> 51      jmp dword CODE_SELECTOR:protect_mode_begin
52
53      ;16位的描述符选择子: 32位偏移
54      ;清流水线并串行化处理器

Remote Thread 1 In: output_bootloader_tag

breakpoint 4, output_bootloader_tag () at bootloader.asm:45
gdb) layout regs
gdb) fs src
focus set to src window.
gdb) fs cmd
focus set to cmd window.
gdb) b bootloader.asm:48
breakpoint 5 at 0x7c97: bootloader.asm:48. (2 locations)
gdb) c
continuing.

breakpoint 5, output_bootloader_tag () at bootloader.asm:48
gdb) s
gdb) █
```

4.远跳转，进入保护模式。

jmp指令将CODE_SELECTOR(0x20)送入cs，将protect_mode_begin +
LOADER_START_ADDRESS送入eip，进入保护模式


```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Register group: general
eax      0x11      17      ecx
edx      0x80      128     ebx
esp      0x7c00    0x7c00    ebp
esi      0x7eec    32492    edi
eip      0x7ea2    0x7ea2    <protect_mode_begin>  eflags
cs       0x20      32      ss
ds       0x0       0       es
fs       0x0       0       gs

--bootloader.asm--
53      ;16位的描述符选择子：32位偏移
54      ;清流水线并串行化处理器
55      [bits 32]
56      protect_mode_begin:
57
> 58      mov eax, DATA_SELECTOR      ;加载数据段(0..4GB)选择子
59      mov ds, eax
60      mov es, eax
61      mov eax, STACK_SELECTOR
62      mov ss, eax
63      mov eax, VIDEO_SELECTOR
64      mov gs, eax
65

remote Thread 1 In: protect mode begin
Breakpoint 4, output_bootloader_tag () at bootloader.asm:45
(gdb) layout regs
(gdb) fs src
Focus set to src window.
(gdb) fs cmd
Focus set to cmd window.
(gdb) b bootloader.asm:48
Breakpoint 5 at 0x7c97: bootloader.asm:48. (2 locations)
(gdb) c
Continuing.

Breakpoint 5, output_bootloader_tag () at bootloader.asm:48
(gdb) s
(gdb) s
(gdb) █
```

5.保护模式状态

Register group: general					
eax	0x18	24	ecx	0x0	0
edx	0x80	128	ebx	0x1c	28
esp	0x7c00	0x7c00	ebp	0x0	0x0
esi	0x7eec	32492	edi	0x0	0
ebp	0x7eb9	0x7eb9 <protect mode begin+23>	eflags	0x6	[PF]
cs	0x20	32	ss	0x10	16
ds	0x8	8	es	0x8	8
fs	0x0	0	gs	0x18	24

```
59    mov ds, eax
60    mov es, eax
61    mov eax, STACK_SELECTOR
62    mov ss, eax
63    mov eax, VIDEO_SELECTOR
64    mov gs, eax
65
> 66    mov ecx, protect mode tag end - protect mode tag
67    mov ebx, 80 * 2
68    mov esi, protect_mode_tag
69    mov ah, 0x3
70    output_protect_mode_tag:
71        mov al, [esi]
```

```
remote Thread 1 In: protect mode begin
(gdb) fs regs
Focus set to regs window.
(gdb)
```

实验任务四

要求：在进入保护模式后，按照如下要求，编写并执行一个自己定义的**32**位汇编程序，要求简单说一说你的实现思路，并提供结果截图。使用两种不同的自定义颜色和一个自定义的起始位置(x,y)，使得**bootloader**加载后，在显示屏坐标(x,y)处开始输出自己的学号+姓名拼音首字母缩写，要求相邻字符前景色和背景色必须是相互对调的。公告图片中提供了学号为**21307233**，姓名为宋小宝，自定义位置**(12,12)**的输出样式，仅供参考

1.清屏(实模式)

```
;清屏
mov ah,0x00
mov ax,3
int 0x10
```

2.进入保护模式(具体步骤看实验任务三)

3.加载段选择子

```
protect_mode_begin:

mov  eax, DATA_SELECTOR           ;加载数据段(0..4GB)选择子
mov  ds,  eax
mov  es,  eax
mov  eax, STACK_SELECTOR
mov  ss,  eax
mov  eax, VIDEO_SELECTOR
mov  gs,  eax
```

4.输出string

```
mov  ecx, my_string_end - my_string
mov  ebx, 0x366           ;使输出位于中间
mov  esi, my_string
output_my_string:
    mov  ah, 0x15         ;确定颜色
    mov  al, [esi]
    mov  word[gs:bx], ax
    inc  esi
    add  ebx,2
    mov  ah, 0x51         ;颜色反转
    mov  al, [esi]
    mov  word[gs:bx], ax
    inc  esi
    add  ebx,2
    sub  ecx,1           ;每次循环ecx减2
    loop output_my_string
```



5. 总结(对实验过程中遇到的问题进行总结，可以提出对实验设置的改进意见)

学会如何使用qemu+gdb来debug,并使用makefile来简化命令行的输入 以下是makefile的使用

```
NAME=mbr
NAME1=mybtld
build:
    @nasm -g -f elf32 $(NAME).asm -o $(NAME).o
    @ld -o $(NAME).symbol -melf_i386 -N $(NAME).o -Ttext 0x7c00
    @ld -o $(NAME).bin -melf_i386 -N $(NAME).o -Ttext 0x7c00 --oformat binary
    @nasm -g -f elf32 $(NAME1).asm -o $(NAME1).o
    @ld -o $(NAME1).symbol -melf_i386 -N $(NAME1).o -Ttext 0x7e00
    @ld -o $(NAME1).bin -melf_i386 -N $(NAME1).o -Ttext 0x7e00 --oformat binary
    @dd if=$(NAME).bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
    @dd if=$(NAME1).bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
debug:
    @qemu-system-i386 -s -S -hda hd.img -serial null -parallel stdio &
    @sleep 1
    @gnome-terminal -e "gdb -tui -q -x gdbinit"
run:
    qemu-system-i386 -hda hd.img -serial null -parallel stdio
clean:
    rm -fr *.bin *.o
```

6. 参考资料清单

- LBA向CHS模式的转换。[https://blog.csdn.net/G_Spider/article/details/6906184]
- int 13h中断。[<https://blog.csdn.net/brainkick/article/details/7583727>]