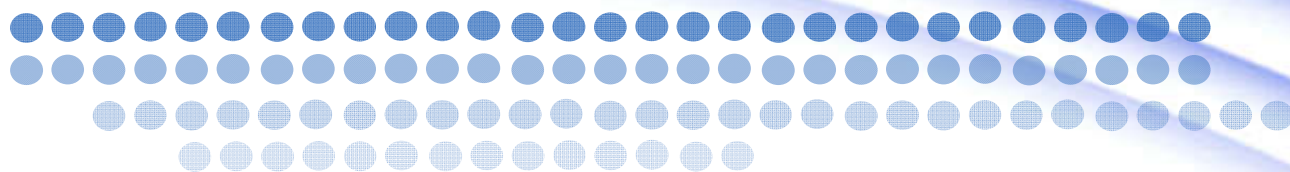


Operating System Principle, OS



# 《操作系统原理》

## 2021级.课程设计

教师：邹德清/李珍/慕冬亮/李志/苏曙光

华中科技大学网安学院

2023年10月-2024年03月

# 课设目的

## ● 课程设计目的

- 掌握OS直接依赖的保护模式的硬件机制和工作原理

- ◆ 无OS支持下直接在保护模式下创建多任务和切换任务

- 掌握保护模式下地址映射机制和设计相应数据结构

- ◆ 段机制和页机制

- 理解保护模式下任务的创建和切换过程

- 理解和实现优先数任务调度策略

- 编程技能培养

- ◆ 掌握保护模式的初始化

- ◆ 掌握段机制的实现

- ◆ 掌握页机制的实现

- ◆ 掌握任务的定义和任务切换

- ◆ 掌握中断程序设计

# 课设任务

## ● 课设任务

- 启动保护模式，创建多个不同优先级的任务（每个任务死循环在屏幕输出不同字符串）。所有任务在时钟（周期50ms，可调）驱动下进行调度，调度策略采用“**优先数进程调度策略**”。
- 示例：4个任务（优先级分别为16,10,8,6）各自输出：**VERY** **LOVE** **HUST** **MRSU** 字符串，每个字符串的持续显示时间长度有差异（体现了优先级的差异）【示例视频见群附件-课设效果演示视频.mp4】。可以观察到**VERY**显示时间最长，**MRSU**最短，**LOVE**，**HUST**居中。



# 课设内容（四级，高级内容包含低级内容）和分值

## ● 内容1（≤ 70分）

- 实现保护模式初始化程序，能将CPU带入保护模式，实现不同权限级别的代码跳转。在屏幕输出相应的字符串指示程序工作情况状态。

- ◆ 代码至少含有GDT,LDT,TSS,GATE等数据结构

## ● 内容2（≤ 80分）

- 定义2个任务A和B（含页目录和页表），并实现任务A切换到任务B。

- ◆ 任务A和B都不是死循环，A先运行，然后切换到B（此后不需要切换回A）

- ◆ 任务A和B的功能代码（即输出字符串的代码）要求在用户态（模仿应用程序的态）。

- ◆ 为设计页表简单，线性地址与物理地址可一一对应，但遇到特定地址可自行设计映射。

## ● 内容3（≤ 95分）

- 2个任务A和B在时钟中断服务程序中实现A和B轮流切换（前述内容2不要求相互切换）。

- ◆ 任务A和B都是死循环。

- ◆ 时钟中断服务程序实现切换任务

## ● 内容4（≤ 100分）

- 定义多个任务，每个任务的优先数不同，在时钟中断服务程序中，采用“优先数调度算法”实现多个任务的切换。

- ◆ 每个任务都是死循环，循环输出字符串

- ◆ 每个任务字符显示时长与优先级一致（优先数大，运行机会多，显示时长就大）。

- ◆ 时钟中断服务程序（选择任务，切换任务）

# 必需的预备知识

## ● 预备知识

- X86的保护模式知识（课件“Intel内存管理”节 + 教材7.8节 + baidu“保护模式”）
- Bochs虚拟机使用（回顾实验1.1）
- NASM汇编（回顾实验1.1）
- 编程实践和指南：于渊《自己动手写操作系统》前3章
- 群中附件“于渊配套书的源代码和可用的**freedos**映像以及**bochs**配置文件”





# 课设要求提交的文档和考核方式

- 课设报告（参照模板）（**70%**）（如果线上进行，此环节为**80%**）
  - （1）课设报告的纸质版（独立完成，老师查重，内容雷同都记**0分**）
  - （2）课设报告的电子版(E-Mail给老师指定邮箱)
  - （3）课设的源工程 (E-Mail给老师指定邮箱)
  - （4）录制**3-5分钟**小视频(E-Mail给老师指定邮箱)。
    - ◆ 视频的要求：展示开发和运行环境的配置，展示程序运行过程，突显运行效果，按序展示源代码每个文件，每一行都要能看到。
    - ◆ 备注1：报告中，原理解释清晰、代码独特性强，都会给高分。
    - ◆ 备注2：报告中，图文清晰，排版美观，会额外加**5分**。
- 当堂检查（**10%**）（如果线上进行，此环节取消）
  - ◆ 课设最后**25分钟**内，检查完成的内容：四级进度和代码质量。
- 在线回答问题（**20%**）
  - 课设最后**20分钟**，在线完成**20道**课设相关的选择或判断题，当堂提交。

# 内容1-3的程序参考步骤和思路

## ● 大致步骤和思路

- (0) 安装好Bochs虚拟机，内存设置为16M即可
- (1) 定义段和描述符表
- (2) 初始化描述符表
- (3) 初始化选择子
- (4) 定义16位的各功能段
- (5) 定义32位的各功能段，包括LDT段
- (6) 进入保护模式
- (7) 初始化页表：可以简单地做线性映射
- (8) 定义两个任务：两个TSS，两个LDT，两个页表
- (9) 利用时钟中断切换两个任务
- (10) 任务可以是简单输出字符“HUST”或“MRSU”

于渊参考书的：  
ch3/h/pmtest8.asm

# 内容1-3的程序参考步骤

## ● 大致步骤和思路

- 设置bochs虚拟机合适的参数（如：内存32M，A盘映像）

- 将X86裸机带入保护模式

- 定义两套页表

于渊参考书的：  
ch3/h/pmtest8.asm

- ◆ 主体相同，但对某个特定线性地址映射到不同物理地址。

- ◆ 特定线性地址需要程序员事先特别安排/仔细设计

- ◆ 两个不同物理地址也需要程序员事先特别安排/仔细设计

- ◆ 两个不同物理地址事先存放有不同函数FuncA和FuncB

- 在特定的线性地址上执行相同的函数，但实际执行了不同的函数（因为地址映射的结果不同）

- 参考《自己动手写操作系统》第3章pmtest 6 / 7.asm



●此页后面的内容自行阅读

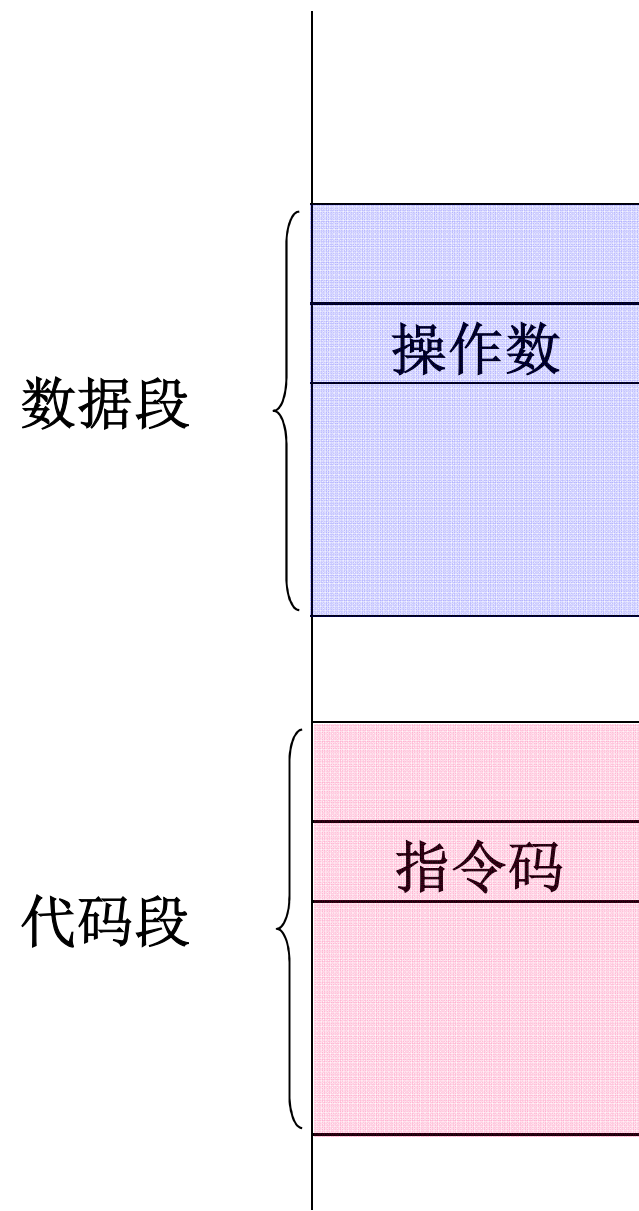
## 课设任务专题讲座（约30分钟，同学们可选参加，不考勤）

- 老师准备在**开学前3-4天**通过**专题讲座**的形式介绍的课设内容，以帮助理解课设内容有困难或对课设预备知识理解有困难的同学。
- **专题讲座**时间请关注群里通知。
- **专题讲座**是非正式的，**不是上课**，仅仅是课后交流。
- **专题讲座**不考勤，自由参加，自由进出课堂。
- **专题讲座**时长大约**30**分钟。



# 段与段描述符 (Descriptor)

- 段
  - 一段连续内存
- 段描述符 **Descriptor**
  - 描述段的属性，8字节
    - ◆ 段基址
    - ◆ 段界限
    - ◆ 段属性
      - 段类型
      - 访问该段所需最小特权级
      - 是否在内存
      - ...



# 段描述符 (Descriptor)

## ● 段描述符

■ 段基址 (32位) : 段限长 (20位)

■ 段属性 (12位)

字节7	字节6	字节5	字节4	字节3	字节2	字节1	字节0
Base1 (31...24)	Attribute		Base2 (23...0)			Limit2 (15...0)	

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
G	D	0	AVL	Limit1 (19...16)				P	DPL		S	TYPE			

# 描述符表 (Descriptor Table)

- 全局描述符表GDT: **Global Descriptor Table**
  - GDT: 包含所有进程可用的段的描述符。
  - 系统仅1个GDT
- 局部描述符表LDT: **Local Descriptor Table**
  - LDT: 与特定进程有关的描述符
  - 每个进程有自己的LDT。
- 中断描述符表IDT: **Interrupt Descriptor Table**
  - 类似中断向量表
  - 包含中断服务程序段的描述符 (中断门/异常门/任务门)
  - 系统仅1个IDT

# GDT的例子

```
1 gdt: ; GDT表
2 ; 第1个描述符不用。
3 .quad 0x0000000000000000
4 ; 第2个是内核代码段描述符。其选择符是0x08
5 .quad 0x00c09a000000007ff
6 ; 第3个是内核数据段描述符。其选择符是0x10
7 .quad 0x00c092000000007ff
8 ; 第4个是显示内存段描述符。其选择符是0x18
9 .quad 0x00c0920b80000002
10 ; 第5个是TSS0段的描述符。其选择符是0x20
11 .word 0x68, tss0, 0xe900, 0x0
12 ; 第6个是LDT0段的描述符。其选择符是0x28
13 .word 0x40, ldt0, 0xe200, 0x0
14 ; 第7个是TSS1段的描述符。其选择符是0x30
15 .word 0x68, tss1, 0xe900, 0x0
16 ; 第8个是LDT1段的描述符。其选择符是0x38
17 .word 0x40, ldt1, 0xe200, 0x0
```

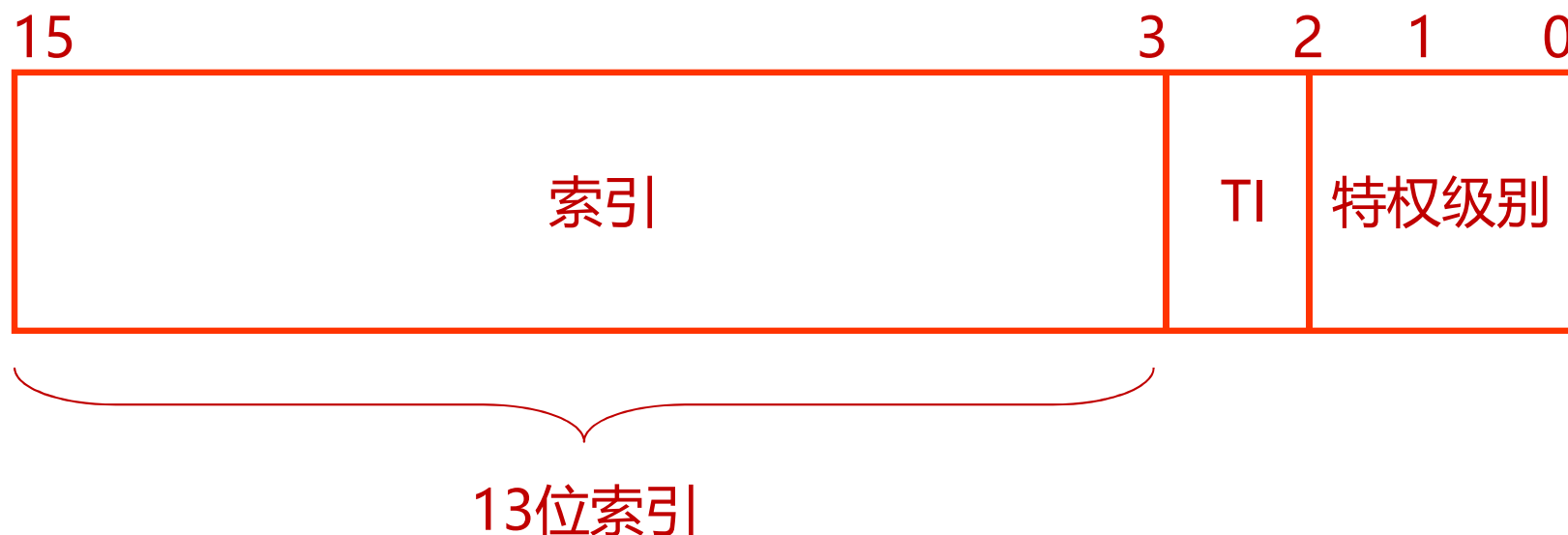


## ● 典型的段描述符

- 数据段/代码段描述符
- 调用门(Call Gate)描述符
- 任务门(Task Gate)描述符
- 中断门(Interrupt Gate)描述符
- 陷阱门(Trap Gate)描述符
- 局部描述符表(LDT)描述符
- 任务状态段TSS描述符

# 选择子 (Selector)

- 选择子是索引，用于选择**GDT/LDT**中的某个描述符。
- 存放在段寄存器中。
  - 索引域 (INDEX) : 13位
  - TI域 (Table Indicator) : 1位
  - 特权级别域 (Request Privilege Level) : 2位



# 保护模式的含义

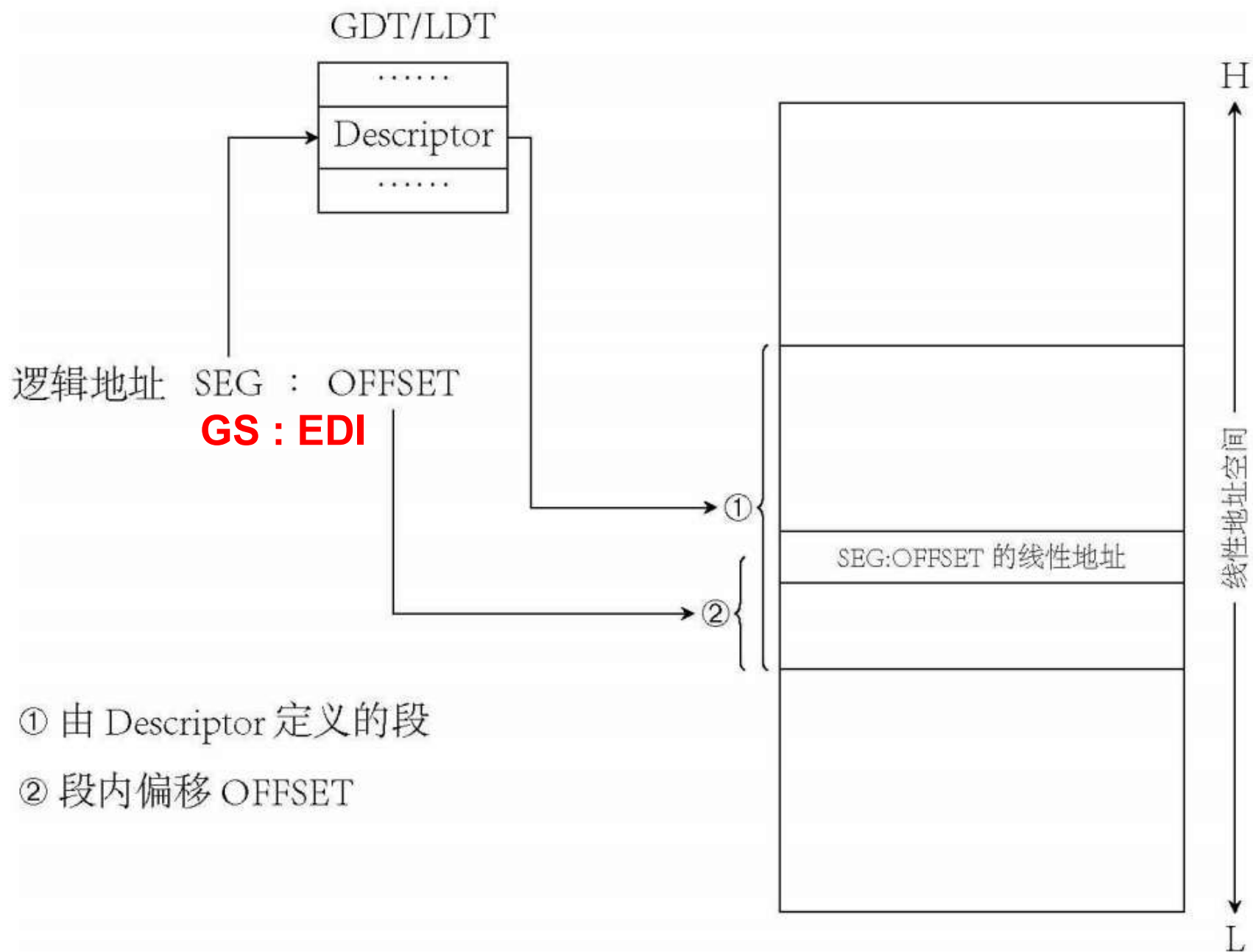


图3.6 段式寻址示意图

# 进入保护模式的步骤

- 1 .创建一个有效的全局描述表
  - 初始化/完善相应的段描述符/选择子
- 2 .创建一个有效的中断描述表
  - 初始化/完善相应的段描述符/选择子
- 3 .关中断
- 4 .用**GDTR**指向创建的全局描述符
- 5 .用**IDIR**指向创建的中断描述符
- 6 . **MSW**中的**PE**位置1
- 7 .跳入到保护模式下的代码/ **JMP CS':EIP'**
- 8 .保护模式下代码
  - 装载**DS**和**SS**的选择子
  - 设置保护模式下堆栈段
  - .....

# 进入保护模式的步骤

```
1 ;; 文件:EnterProtectMode.asm
2 ;; 工具:UltraEdit14.12编辑, Nasm2.02汇编
3 ;; 作用:由实模式进入保护模式的一种方法
4 ;; 备注:1.44M 512bits/80sec软盘启动
5 ;;=====
6 [BITS 16] ;编译成16位指令
7 [ORG 0x7C00]
8 cli ;关闭中断, 保证引导程序在执行时不被打扰
9 xor ax,ax
10 mov ds,ax
11
12 lgdt[GDTR_Value] ;加载GDTR:将GDT基址及大小装入GDTR=limit(16)+base(32)
13 mov eax,CR0
14 or eax,1 ;设置eax的第0位:PE位,
15 mov CR0,eax ;置PE位, 此行之后进入保护模式
16 jmp 08h:GoIntoProtectMode ;08h:跳过GDT第一个段空段(00h-07h),即08h
17
```

# 进入保护模式

```
18 [BITS 32]           ;编译成32位指令
19 GoIntoProtectMode: ;因为要进保护模式，所以对DS, CS, ES, SS, FS, GS重写
20     mov ax, 10h      ;10h:GDT(00h-07h):空,GDT(08-0fh)代码段,GDT(10h-17h)数据段
21     mov ds, ax       ;
22     mov ss, ax       ;堆栈段与数据相同
23     mov esp, 090000h ;
24     ;保护模式下不能直接使用BIOS中断，显示要是向显存缓冲里直接写入
25     ;显存位于:0xA0000---0xbffff, 帧缓冲位于0xb8000处
26     ;字符2个字节：字节1代表ASCII，字节2属性：前景色/背景色/闪烁
27     mov byte [ds:0B8000h], 'I'
28     mov byte [ds:0B8001h], 1ah
29     mov byte [ds:0B8002h], 'S'
30     mov byte [ds:0B8003h], 9bh
31     mov byte [ds:0B8004h], '1'
32     mov byte [ds:0B8005h], 1ch
33     mov byte [ds:0B8006h], '8'
34     mov byte [ds:0B8007h], 9dh
35     mov byte [ds:0B8008h], '!'
36     mov byte [ds:0B8009h], 1eh
37 STOP:
38     jmp STOP
```



# 进入保护模式

```
40 GDT:      ;填写GDT, 1个空段, 1个代码段, 1个数据段
41 GDT_Null:      ;填写GDT中的NULL段描述符, Intel保留的区域, 用零填充
42     DD 0      ;共64位的0
43     DD 0
44
```

# 进入保护模式

```
45 GDT_Code:      ;填写GDT中的代码段描述符
46     DW 0ffffh   ;填充limit(15-0),共16位1
47     DW 0        ;基址为0
48     DB 0        ;十六位的低八位: 仍是基址, 填0
49     DB 10011010B ;十六位的高八位: 低到高:A, R/W, ED/C, E, S, DPL, P
50                 ;A是访问标志, 由CPU在第一次访问时设置, 置0;
51                 ;R/W置为1使段可读;
52                 ;ED/C顺从性, 如置1, 低优先级代码可跳转到或调用该段。
53                 ;E置为1, 表示描述符描述的是代码段; E置为0, 表示数据段
54                 ;S表示该段是代码段或数据段, 置为1; 系统置0
55                 ;DPL表示优先级, 由于是引导程序, 所以要把优先级设为00;
56                 ;P设置为1, 段有有效的基址和界限。没有定义描述则置0
57     DB 11001111B ;十六位的低八位: 四位偏移, AV, 0, D, G
58                 ;首先4位偏移量, 设置为0Fh;
59                 ;AV=1表示segment is available, 此处忽略该位, 设为0;
60                 ;Intel保留了一位必须设为0;
61                 ;D表示大小位, 置为1, 它告诉CPU使用32位代码而不是16位代码;
62                 ;G表示粒度, 如果G=0, 则Limit所表示的段偏移是00000H-FFFFFH,
63                 ;如果G=1, 则Limit表示的段偏移是00000XXXH-FFFFFXXXH,
64                 ;即Limit所表示的段偏移实际上是它的值再乘以4K。此处设置G=1。
65     DB 0        ;十六位的高八位: 基址, 全置0
```

# 进入保护模式

```
67 GDT_Data:      ;填写GDT中的数据段描述符
68     dw 0ffffh   ;填充limit(15-0),共16位1
69     dw 0         ;基址为0
70     DB 0         ;十六位的低八位: 仍是基址, 填0
71     DB 10010010B ;十六位的高八位: 低到高:A, R/W, ED/C, E, S, DPL, P
72                     ;低位第4位=0, 表示描述符描述的是数据段, 上面代码段置的是1
73     DB 11001111B ;十六位的低八位: 四位偏移, AV, 0, D, G
74     DB 0         ;十六位的高八位: 基址, 全置0

75 GDT_End:        ;由于在lgdt的时候需要把GDT的地址和大小加载到GDTR中,
76                 ;本条指令GDT的结束, 是为了计算GDT的大小

78 GDTR_Value:     ;GDT的描述符, 被lgdt加载到GDTR=基址+大小
79     DW GDT_End - GDT - 1 ;计算GDT的大小, 注意减1
80     DD GDT        ;基址
```

# 进入保护模式

```
67 GDT_Data:      ;填写GDT中的数据段描述符
68     dw 0ffffh   ;填充limit(15-0),共16位1
69     dw 0         ;基址为0
70     DB 0         ;十六位的低八位: 仍是基址, 填0
71     DB 10010010B ;十六位的高八位: 低到高:A, R/W, ED/C, E, S, DPL, P
72                     ;低位第4位=0, 表示描述符描述的是数据段, 代码段置的是1
73     DB 11001111B ;十六位的低八位: 四位偏移, AV, 0, D, G
74     DB 0         ;十六位的高八位: 基址, 全置0
75 GDT_End:        ;由于在lgdt的时候需要把GDT的地址和大小加载到GDTR中,
76                     ;本条指令GDT的结束, 是为了计算GDT的大小
77
78 GDTR_Value:      ;GDT的描述符, 被lgdt加载到GDTR=基址+大小
79     DW GDT_End - GDT - 1 ;计算GDT的大小, 注意减1
80     DD GDT         ;基址
81
82     times 510-($-$$) db 0
83     DW 0AA55h
```

- 任务的一般结构

- 每个任务的上下文信息

- ◆ TSS

- ◆ TSS描述符存储在GDT中

- 每个任务的LDT

- ◆ LDT

- ◆ LDT描述符被加载到GDT

- 例：建立两个任务

# 例：建立两个任务

- 数据结构

- GDT

- ◆ 代码段描述符、数据段描述符、显示的描述符、TSS0描述符、LDT0描述符、TSS1描述符、LDT1描述符

- LDT0

- ◆ 数据段描述符，代码段描述符

- LDT1

- ◆ 数据段描述符，代码段描述符

- TSS0

- TSS1

- 堆栈支持



# 例：建立两个任务

- 内存分布

- IDT

- GDT

- LDT0+TSS0+任务0的核心堆栈

- LDT1+TSS1+任务1的核心堆栈

- TASK0

- TASK1

- 任务0的用户栈

- 任务0的用户栈

- 任务调度

- 时钟发生器

- IDT（异常或中断处理过程），IDTR

```

376 tss0:
377     dd 0
378     dd stack0_krn_ptr, 0x10 ;任务0的核心态使用的堆栈,就紧跟在 TSS0 的后面!
379     dd 0,0
380     dd 0,0
381     dd 0
382     dd task0                ;确保第一次切换到任务0时EIP从这里取值,故从task0处开始运行,
383     |                       ;因为任务0的基址为 0x10000,和核心一样,不指定这个的话,
384     |                       ;第一次切换进来就会跑去执行 0x10000处的核心代码了,
385     |                       ;除非ldt0中的代码段描述符中的基址改成 task0 处的线性地址,
386     |                       ;那这里就可以设为0.
387     dd 0x200
388     dd 0,0,0,0
389     dd stack0_ptr,0,0,0
390     dd 0x17,0x0F,0x17,0x17,0x17,0x17
391     dd LDT0_SEL
392     dd 0x08000000
393     times 128 dd 0
394 stack0_krn_ptr:
395     dd 0

```

```
403 tss1:
404     dd 0
405     dd stack1_krn_ptr, 0x10
406     dd 0, 0
407     dd 0, 0
408     dd 0
409     dd task1
410     dd 0x200
411     dd 0, 0, 0, 0
412     dd stack1_ptr, 0, 0, 0
413     dd 0x17, 0x0F, 0x17, 0x17, 0x17, 0x17
414     dd LDT1_SEL
415     dd 0x08000000
416     times 128 dd 0
417 stack1_krn_ptr:
418     dd 0
```

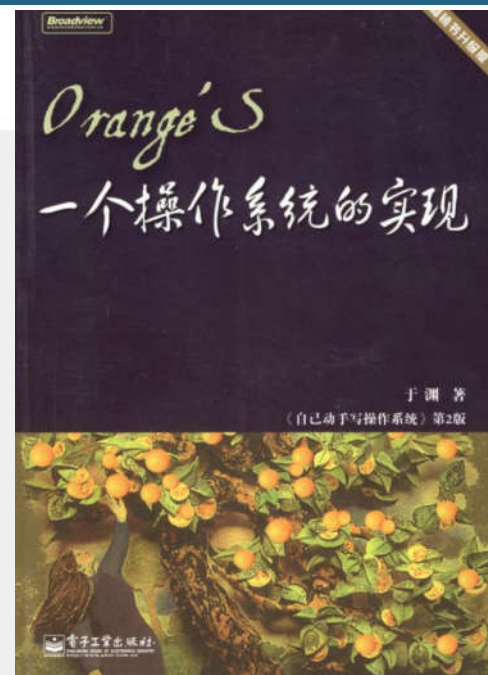
```
420 task0:
421     mov eax,0x17
422     mov ds,ax
423     mov al,'1' HUST
424     int 0x80
425     mov ecx,0xFFF
426 x3:
427     loop x3
428     jmp task0
429
430     times 128 dd 0
431 stack0_ptr:
432     dd 0
```

```
434 task1:
435     mov eax,0x17
436     mov ds,ax
437     mov al,'0' -SSE
438     int 0x80
439     mov ecx,0xFFF
440 x4:
441     loop x4
442     jmp task1
443
444     times 128 dd 0
445 stack1_ptr:
446     dd 0
```

# 源代码代码分析：第三章 pmtest1.asm

代码3.1 chapter3/a/pmtest1.asm

```
1 ; =====
2 ; pmtest1.asm
3 ; 编译方法: nasm pmtest1.asm -o pmtest1.bin
4 ; =====
5
6 %include "pm.inc" ; 常量, 宏, 以及一些说明
7
8 org 07c00h
9 jmp LABEL_BEGIN
10
11 [SECTION .gdt]
12 ; GDT
13 ; 段基址, 段界限, 属性
14 LABEL_GDT: Descriptor 0, 0, 0 ; 空描述符
15 LABEL_DESC_CODE32: Descriptor 0, SegCode32Len - 1, DA_C + DA_32; 非一致代码段
16 LABEL_DESC_VIDEO: Descriptor 0B8000h, 0ffffh, DA_DRW ; 显存首地址
17 ; GDT 结束
18
19 GdtLen equ $ - LABEL_GDT ; GDT长度
20 GdtPtr dw GdtLen - 1 ; GDT界限
21 dd 0 ; GDT基地址
22
23 ; GDT 选择子
24 SelectorCode32 equ LABEL_DESC_CODE32 - LABEL_GDT
25 SelectorVideo equ LABEL_DESC_VIDEO - LABEL_GDT
26 ; END of [SECTION .gdt]
27
28 [SECTION .s16]
```



# 实验环境准备

- 实验环境准备

- 操作系统Linux

- 汇编器nasm

- 虚拟机bochs（内含bximage工具）

- freedos软盘映像文件



## 两个任务切换的效果（视频）

```
V AL, 0x20
T 0x20
V EAX, [currentTask]
P [currentTask]
x2
currentTask
V [currentTask]
P Select
P y2
currentTask
V DWORD [currentTask]
```

```
A:\>b:
B:\>dir

Volume in drive B:
In Protect Mode no
BaseAddrL BaseAddr
00000000hC00000000
0009F000hC00000000
000EB000hC00000000
00100000hC00000000
01FF0000hC00000000
FFFC0000hC00000000
PMTST8 COM
RAM size:01FF0000h
HUST2 COM
TWOTASKS COM
HUSTMRSU COM
12 files
0 dirs
B:\>hustmrsu.com

IPS: 62.343M
```

## ● 准备实验环境的实际步骤

- Ubuntu16.04/Ubuntu20.10

- 在线安装nasm

- 在线安装虚拟机bochs（内含bximage工具）

- 在bochs 官网下载freedos软盘映像文件

  - ◆ 软盘A： 启动盘， 系统盘

- 制作用于存储测试程序的空白软盘映像文件

  - ◆ 软盘B： 用户盘， 存放用户的测试程序

# 准备实验环境的实际步骤

## ● Ubuntu16.04/Ubuntu20.10

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ cat /etc/issue
Ubuntu 16.04.3 LTS \n \l

susg@ThinkPad:~/os/OSDesign/YuYuanBook$ uname -r
4.10.0-28-generic
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

# 准备实验环境的实际步骤

## ● 在线安装nasm

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo apt-get install nasm
[sudo] password for susg:
Reading package lists... Done
Building dependency tree
Reading state information... Done
nasm is already the newest version (2.11.08-1ubuntu0.1).
0 upgraded, 0 newly installed, 0 to remove and 668 not upgraded.
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

# 准备实验环境的实际步骤

- 在线安装nasm

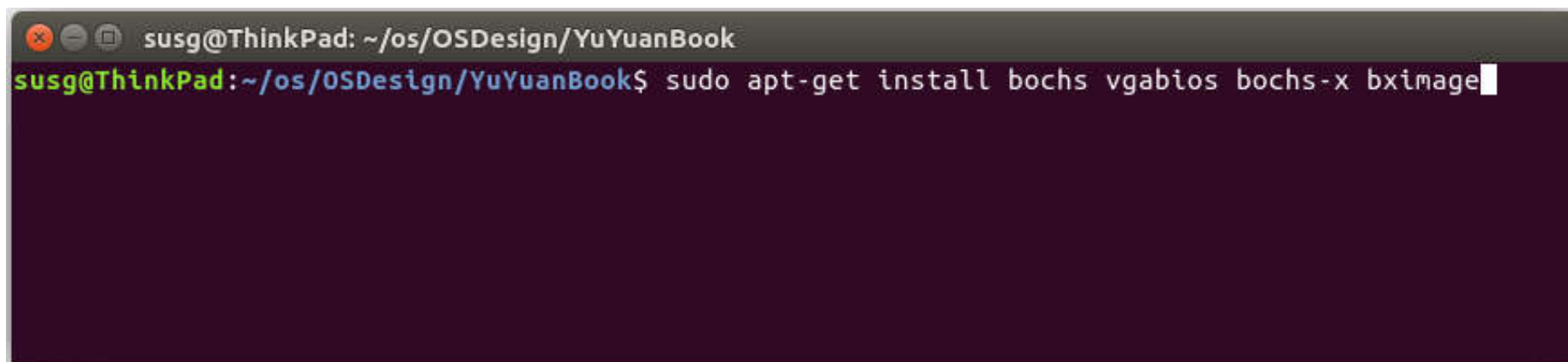
- 检验nasm是否安装正确

- `nasm pmtest1.asm -o pmtest1.com`

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook/pmtestSrc/pmtest1
susg@ThinkPad:~/os/OSDesign/YuYuanBook/pmtestSrc/pmtest1$ ls
pm.inc  pmtest1.asm
susg@ThinkPad:~/os/OSDesign/YuYuanBook/pmtestSrc/pmtest1$ nasm pmtest1.asm -o pmtest1.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook/pmtestSrc/pmtest1$ ls
pm.inc  pmtest1.asm  pmtest1.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook/pmtestSrc/pmtest1$
susg@ThinkPad:~/os/OSDesign/YuYuanBook/pmtestSrc/pmtest1$
```

# 准备实验环境的实际步骤

- 在线安装虚拟机**bochs**（内含**bximage**工具）



```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo apt-get install bochs vgabios bochs-x bximage
```



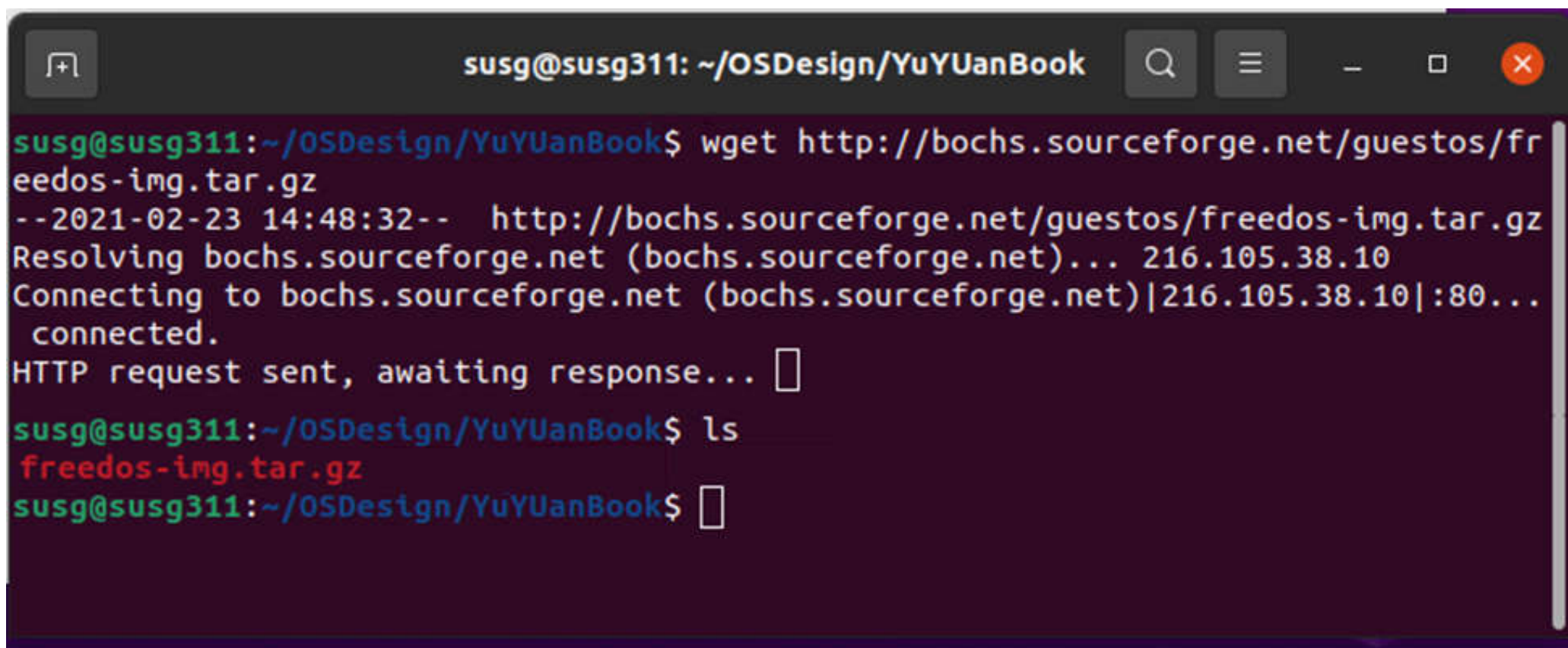
# 准备实验环境的实际步骤

- 在线安装虚拟机**bochs**（内含**bximage**工具）
  - 检查bochs是否安装成功
  - 检查bochs安装目录，为修改bochrc文件备用

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ whereis bochs
bochs: /usr/bin/bochs /usr/lib/bochs /usr/share/bochs /usr/share/man/man1/bochs.1.gz
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

# 准备实验环境的实际步骤

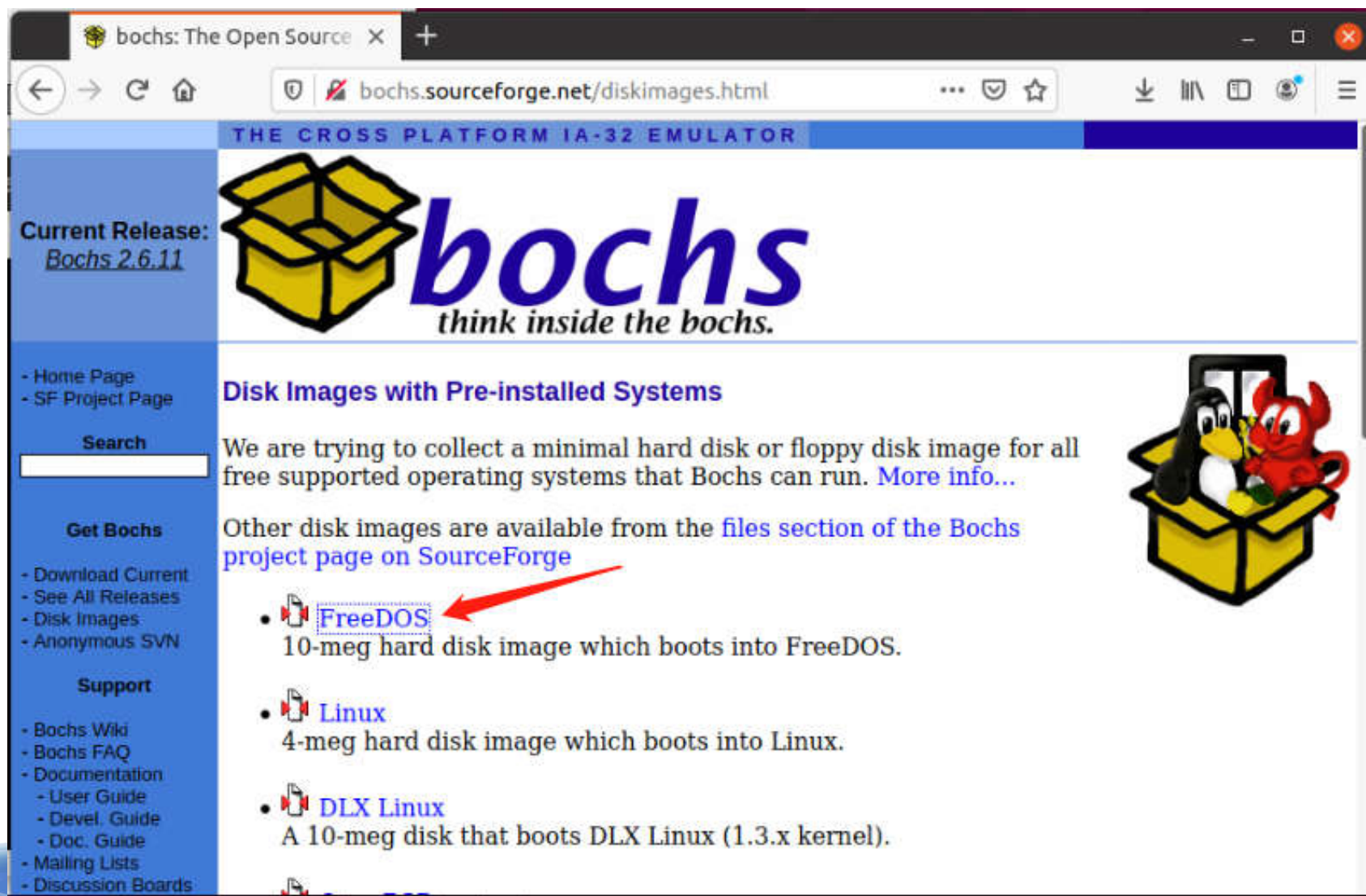
- 在**bochs** 官网下载**freedos**软盘映像文件
  - QQ群中附件也有此映像文件



```
susg@susg311: ~/OSDesign/YuYUanBook
susg@susg311:~/OSDesign/YuYUanBook$ wget http://bochs.sourceforge.net/guestos/freedos-img.tar.gz
--2021-02-23 14:48:32--  http://bochs.sourceforge.net/guestos/freedos-img.tar.gz
Resolving bochs.sourceforge.net (bochs.sourceforge.net)... 216.105.38.10
Connecting to bochs.sourceforge.net (bochs.sourceforge.net)|216.105.38.10|:80...
connected.
HTTP request sent, awaiting response... 
susg@susg311:~/OSDesign/YuYUanBook$ ls
freedos-img.tar.gz
susg@susg311:~/OSDesign/YuYUanBook$
```

# 准备实验环境的实际步骤

## ● 在bochs 官网下载freedos软盘映像文件





# 准备实验环境的实际步骤

- 在**bochs** 官网下载**freedos**软盘映像文件
  - 解压freedos后内含4个文件
  - 仅保留a.img并更名为freedos.img备用
    - ◆ 软盘A: 启动盘, 系统盘

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ ls ./freedos-img/
a.img b.img bochsrc c.img
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ cp ./freedos-img/a.img ./freedos.img
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ cp ./freedos-img/bochsrc ./bochsrc.txt
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ ls
bochsrc.txt freedos-img freedos.img pntestSrc
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

# 准备实验环境的实际步骤

- 制作用于存储测试程序的空白软盘映像文件
  - 软盘B：用户盘，存放用户的测试程序
  - 利用Bochs内含工具bxiimage制作： pmtest.img

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook

susg@ThinkPad:~/os/OSDesign/YuYuanBook$ bxiimage
=====
                        bxiimage
                Disk Image Creation Tool for Bochs
                $Id: bxiimage.c 11315 2012-08-05 18:13:38Z vruppert $
=====

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] fd

Choose the size of floppy disk image to create, in megabytes.
Please type 0.16, 0.18, 0.32, 0.36, 0.72, 1.2, 1.44, 1.68, 1.72, or 2.88.
[1.44]

I will create a floppy image with
cyl=80
heads=2
sectors per track=18
total sectors=2880
total bytes=1474560

What should I name the image?
[a.img] pmtest.img

Writing: [] Done.

I wrote 1474560 bytes to pmtest.img.

The following line should appear in your bochsrc:
floppya: image="pmtest.img", status=inserted
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ ls
bochsrc.txt  freedos-img  freedos.img  pmtest.img  pmtestSrc
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

空白盘，且尚未  
格式化，暂时不  
能用。后面会对  
其格式化。

# 准备实验环境的实际步骤

## ● 配置bochs的运行控制文件bochsrc.txt

```
1 # 文件名: bochsrc.txt
2 # 指定虚拟机的内存大小
3 megs: 32
4 # BIOS ROM 映像文件名
5 romimage: file=/usr/share/bochs/BIOS-bochs-latest
6 # VGA ROM 映像文件名
7 vgaromimage: file=/usr/share/bochs/VGABIOS-lgpl-latest
8 # 指定软盘的映像文件和状态 (是否已经插入)
9 floppyb: 1_44=freedos.img, status=inserted
10 floppyb: 1_44=pmtest.img, status=inserted
11 # 指定启动设备
12 boot: a
13 # 禁用鼠标
14 mouse: enabled=0
```

路径要据实填写。  
# whereis bochs



# 准备实验环境的实际步骤

- 利用 **bochsrc.txt** 测试 **bochs** 能否正常工作？
  - freedos.img（软盘A）+ pmtest.img（软盘B）
  - 格式化 **pmtest.img**（软盘B）（不然后面 **mount** 出错！）

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ bochs -f bochsrc.txt
=====
Bochs x86 Emulator 2.6
Built from SVN snapshot on September 2nd, 2012
=====
000000000000i[      ] using log file bochsout.txt
Next at t=0
(0) [0x00000000ffffff0] f000:fff0 (unk. ctxt): jmp far f000:e05b      ; ea5be000f0
<bochs:1> C
```

```
Bochs x86-64 emulator, http://bochs.sourceforge.net/
A:\>format.exe B:
Reading boot sector...
Cylinder:      0 Head:      0

Saving UNFORMAT information...
Cylinder:      77 Head:      1

Creating file system...
Cylinder:      0 Head:      0

Format operation complete.
A:\>
```

空白盘，格式化。  
format B: 错！  
要用后缀 .exe！

# 准备实验环境的实际步骤

- 将格式化的**pmtest.img**软盘映像挂接到某个目录
  - 便于将来频繁地更新其中的实验程序
  - `mount -o loop ./pmtest.img /mnt/floppyB`

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo mkdir /mnt/floppyB
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo mount -o loop pmtest.img /mnt/floppyB/
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest1/pmtest1.com /mnt/floppyB/pmtest1.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest2/pmtest2.com /mnt/floppyB/pmtest2.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest3/pmtest3.com /mnt/floppyB/pmtest3.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest4/pmtest4.com /mnt/floppyB/pmtest4.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest5/pmtest5.com /mnt/floppyB/pmtest5.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest6/pmtest6.com /mnt/floppyB/pmtest6.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest7/pmtest7.com /mnt/floppyB/pmtest7.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo cp ./pmtestSrc/pmtest8/pmtest8.com /mnt/floppyB/pmtest8.com
susg@ThinkPad:~/os/OSDesign/YuYuanBook$
```

# 课设要做什么

- 任务和内容

- 参考《课设任务》、《课设内容》

- 目标机

- 裸机 | bochs虚拟机（~~ubuntu或windows~~）

- 开发环境

- ubuntu + nasm

- 开发语言

- 汇编程序

- 程序功能

- 两/多个任务在屏幕同一位置持续显示各自字符串。

屏幕/显示缓冲区

H	R	S	U				