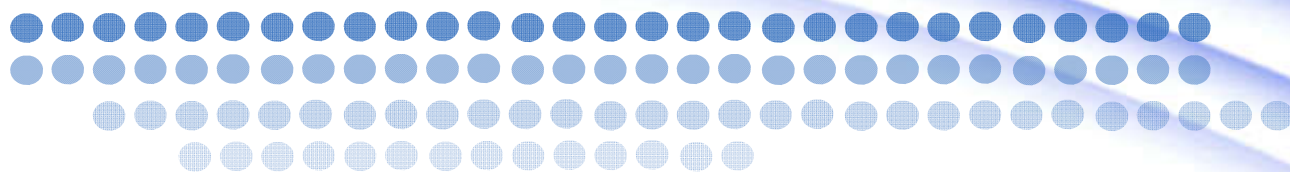


Operating System Principle, OS



《操作系统原理》

2022级.课程设计

教师：邹德清/李珍/慕冬亮/李志/王凯龙/苏曙光

华中科技大学网安学院

2025年02月-2025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

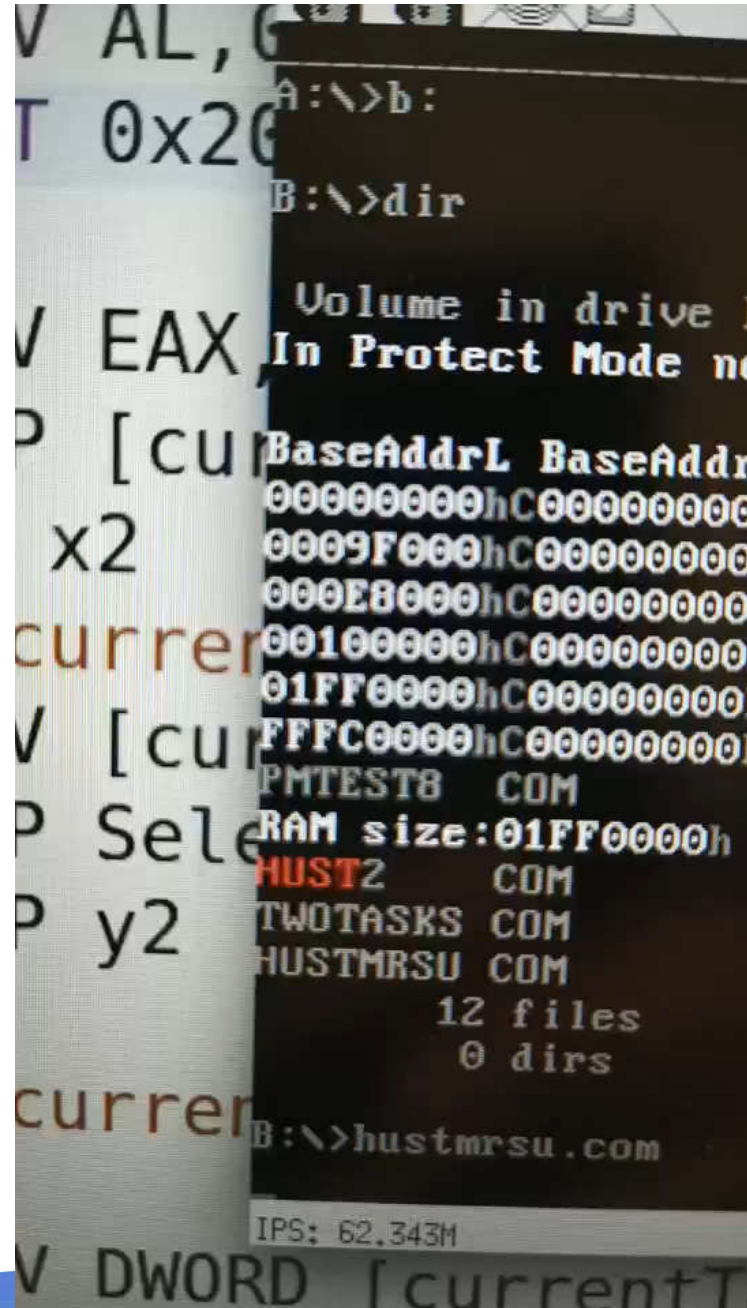
● 课设任务

- 裸机(bochs)中启动保护模式，创建多个不同优先级的任务（每个任务死循环在屏幕相同位置输出不同字符串，（含TSS，LDT，代码，堆栈，页目录/页表等要素））。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）【参考：课设效果演示视频1.mp4】。可以观察到**VERY**显示时间最长，**MRSU**最短，**LOVE**，**HUST**居中。



两个任务切换的效果（视频）

- 参考：课设效果演示视频2.mp4



```
AL, 0x20
T 0x20
V EAX, 0x20
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
PMTEST8 COM
RAM size:01FF0000h
HUST2 COM
TWOTASKS COM
HUSTMRSU COM
12 files
0 dirs

B:\>hustmrsu.com

IPS: 62.343M
```

分级任务（不同级别任务的得分上限不同：70,80,90,100）

● 级别1内容（≤ 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，门等概念

● 级别2内容（≤ 80分）

- 定义2个任务A和B（两套页目录和页表），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（此后不切换）

- ◆ 任务A和B的功能代码（即输出字符串的代码）都是用户态。

- ◆ 线性地址与物理地址可采用一一对应关系，特定地址自行映射。

● 级别3内容（≤ 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。

- ◆ 任务A和B都是死循环。

- ◆ 时钟中断服务程序（切换任务）

● 级别4内容（≤ 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）。

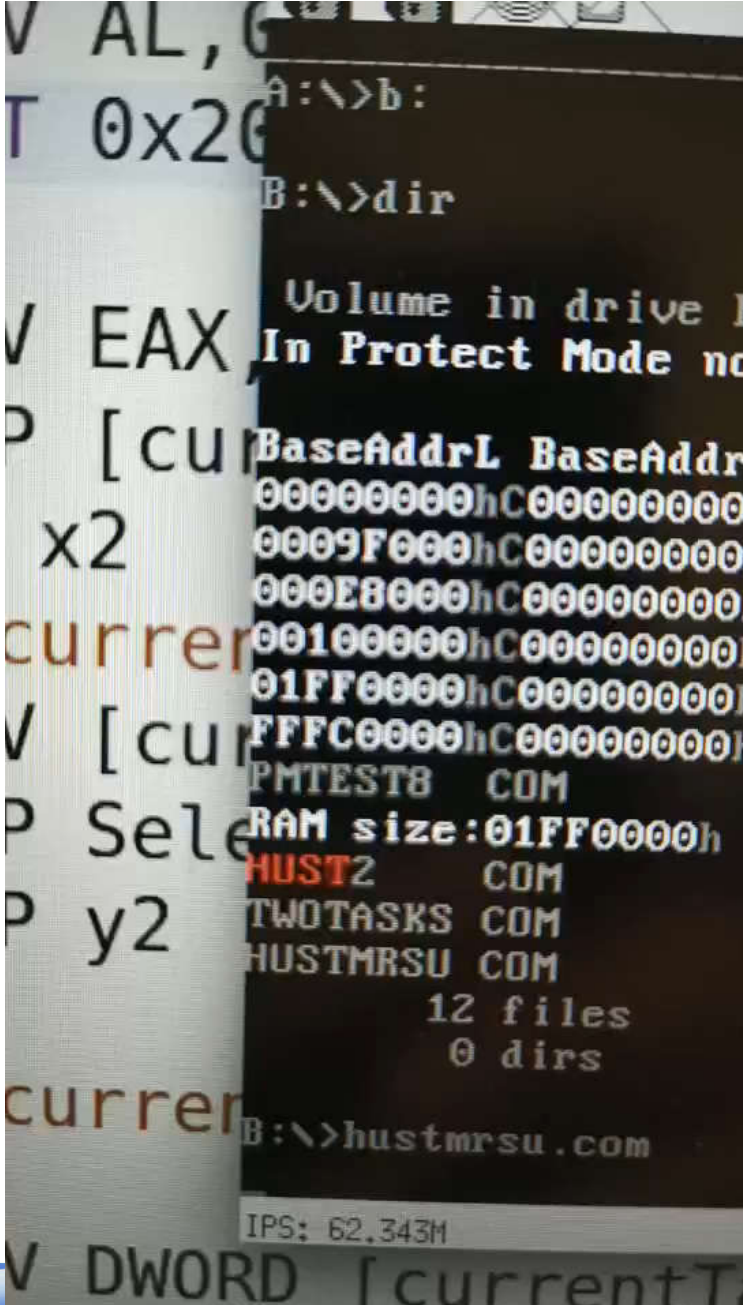
- ◆ 时钟中断服务程序（选择任务，切换任务）

2022级操作系统原理课设任务

- 当场验收活动：第二周周五下午**16:00**开始；
- 在线测验活动：第二周周五下午**17:00**开始.
- 验收的准备工作
 - (1) 每个全程**30秒**。从老师走到你座位开始算起。
 - (2) 任务分内容**1~4**，演示已完成的最高级内容。
 - (3) **只有**在freedos命令上**输入完程序名(含参数)**后，**才可以**举手示意老师来检查，老师来后只按回车键就可运行。

具体要求（级别3内容：两个任务在时钟驱动下交替切换）

- 任务和内容
 - 参考《课设任务》、《课设内容》
- 目标机
 - 裸机 | bochs(ubuntu或windows)
- 开发环境
 - ubuntu + nasm
- 开发语言
 - 汇编程序
- 程序功能 【HUST】 | 【MRSU】
 - 两个任务在**同一位置**持续显示各自字符串。



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

Volume in drive B:
In Protect Mode no

BaseAddrL BaseAddr
00000000hC0000000
0009F000hC0000000
000E8000hC0000000
00100000hC0000000
01FF0000hC0000000
FFFC0000hC0000000
PMTEST8.COM
RAM size:01FF0000h
HUST2.COM
TWOTASKS.COM
HUSTMRSU.COM
12 files
0 dirs

B:\>hustmrsu.com

IPS: 62.343M
V DWORD [currentTask]
```

具体要求（级别3内容：两个任务在时钟驱动下交替切换）

- 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	T				

具体要求

● 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。

TaskA:

Print 'H'

Print 'U'

Print 'S'

Print 'T'

Print 'M'

Print 'R'

Print 'S'

Print 'U'

JMP TaskA

屏幕/显示缓冲区

H	R	S	U				

具体要求

● 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。

TaskA:

Print 'H'

Print 'U'

Print 'S'

Print 'T'

RET

TaskB:

Print 'M'

Print 'R'

Print 'S'

Print 'U'

RET

MAIN:

CALL TaskA

CALL TaskB

JMP MAIN

屏幕/显示缓冲区

M	R	S	U				

具体要求

● 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。

TaskA:

Print 'H'

Print 'U'

Print 'S'

Print 'T'

JMP TaskA

TaskB:

Print 'M'

Print 'R'

Print 'S'

Print 'U'

JMP TaskB

屏幕/显示缓冲区

H	R	S	U				

◆ 多任务/task

◆ 任务切换

◆ 时钟中断服务(50MS): 切换不同任务

◆ 调度算法: 50MS循环轮转 | 优先数 (内容4)

必需的预备知识

● 预备知识

- X86的保护模式知识（课件“i386和Linux存储管理”节 + 教材7.8节 + baidu“保护模式”）
- Bochs虚拟机的使用（虚拟机装在Ubuntu中，不是Windows）
- NASM汇编环境的使用（在Ubuntu中用NASM）
- 编程实践和指南：于渊《一个操作系统的实现》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%）

- （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2-3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，排版美观，图文清晰，会额外加5分。

● 当堂检查（10%）

- 课设最后45分钟内，检查完成的内容，进度和代码质量。
- 能演示级别1~4的内容：分别得分2，4，6，10分

● 在线回答问题（20%）

- 课设最后15分钟在线完成10-20个与课设直接相关的客观题，当堂提交。
- ◆只考与课设直接相关的技术、工具、环境、编程技能和关键理论。

级别1-3的任务的实现步骤和思路的参考

ch3/h/pmtest8.asm

● 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”

```
MOV CR3, PT0_base ; 更换为页表0  
CALL LinearAddrDemo
```

```
MOV CR3, PT1_base ; 更换为页表1  
CALL LinearAddrDemo
```

级别1-3的任务的实现步骤和思路的参考

● 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

ch3/h/pmtest8.asm

- 定义两套页表

 - ◆ 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

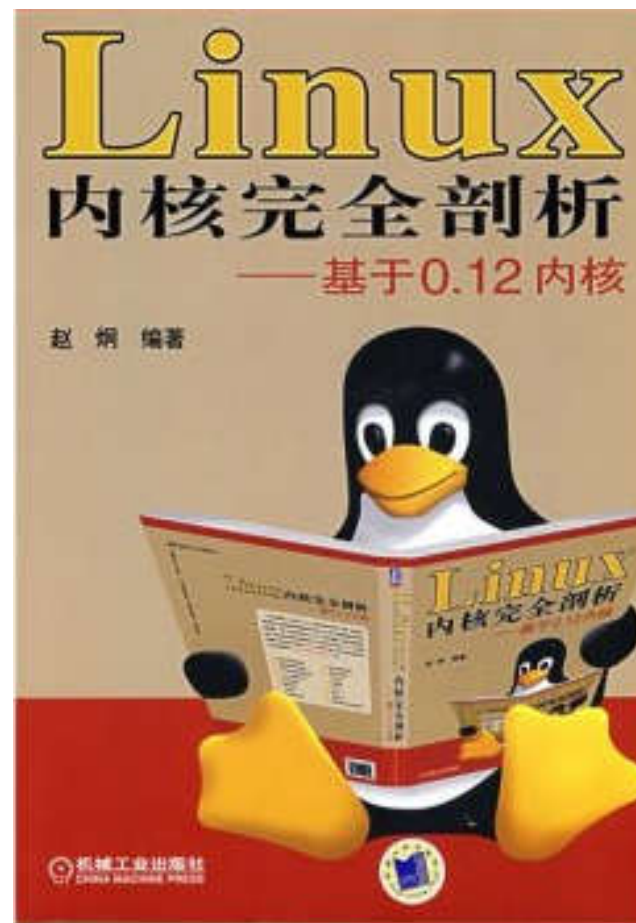
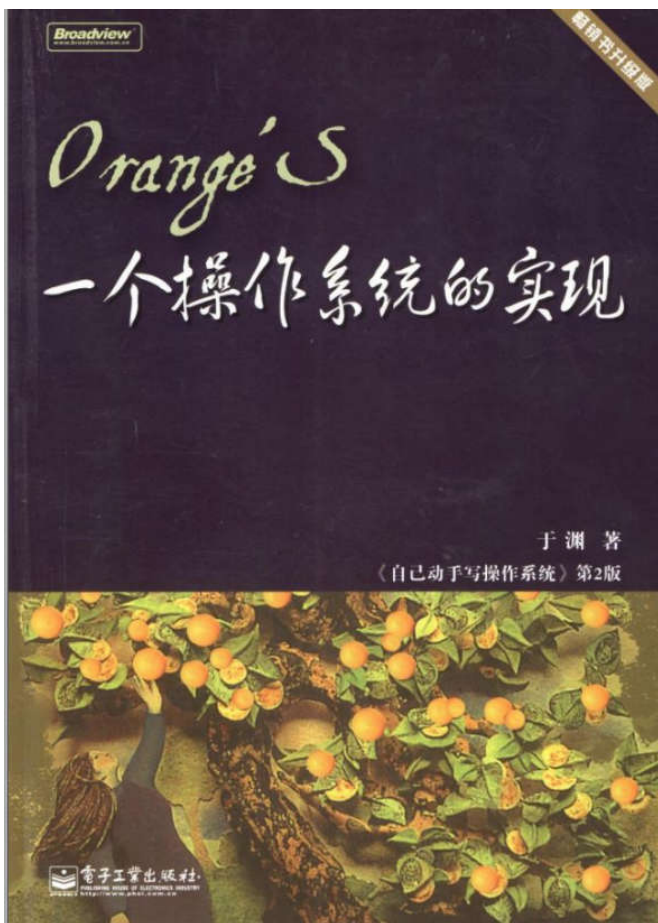
- 在特定的线性地址上执行相同的函数，但实际执行了不同的函数（因为地址映射的结果不同）

- 参考《自己动手写操作系统》第3章pmtest 6 / 7.asm



以下内容来自上课课件

7.5 i386和Linux存储管理



● 保模式寻址

■ 例:逻辑地址 = DS:1000H
= 13H:1000H

■ 段基址=段基址(DS)
=段基址(13H)

■ 描述符、描述表、类型?

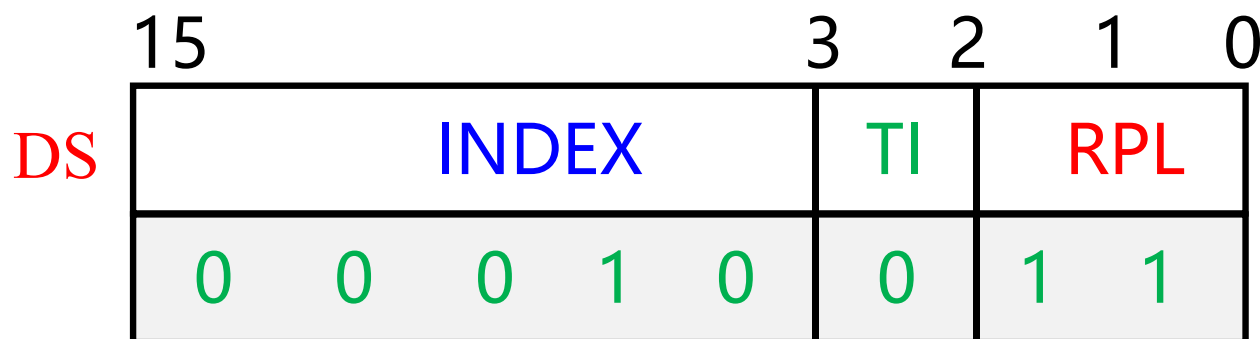
■ DS含义? 段寄存器含义?



INDEX = 2

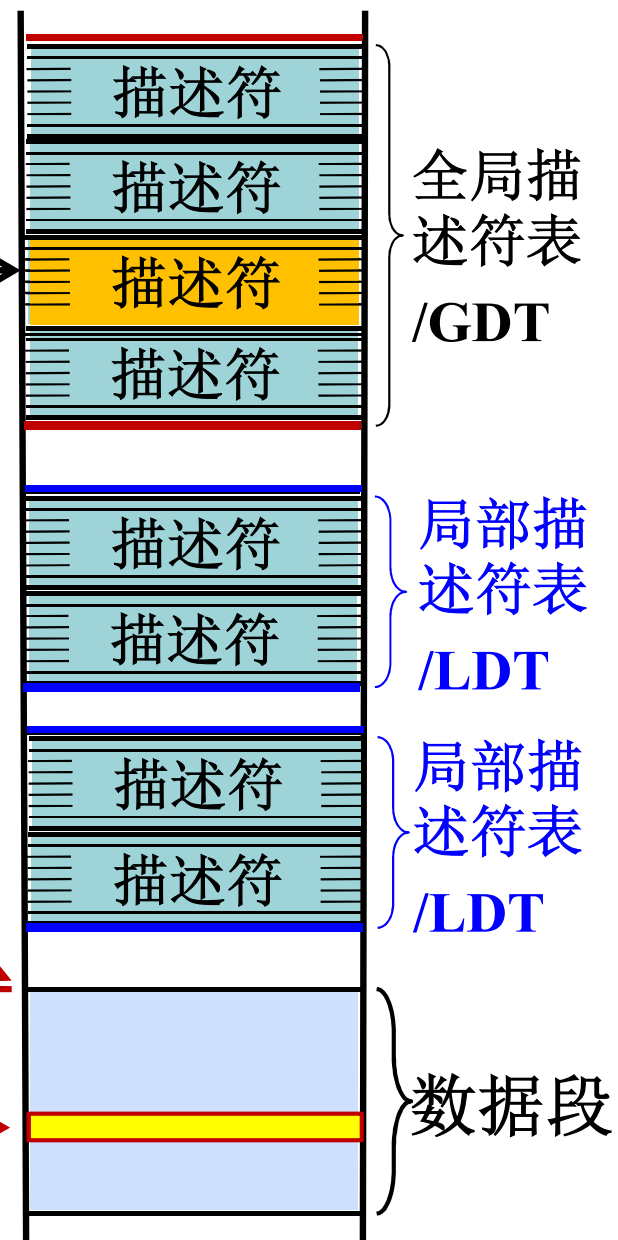
DS

13H



Base

1000H



● 保模式寻址

■ 例:逻辑地址 = DS:1000H
= 13H:1000H

■ 段基址=段基址(DS)
=段基址(13H)

■ 描述符、描述表、类型?

■ DS含义? 段寄存器含义?



INDEX = 2

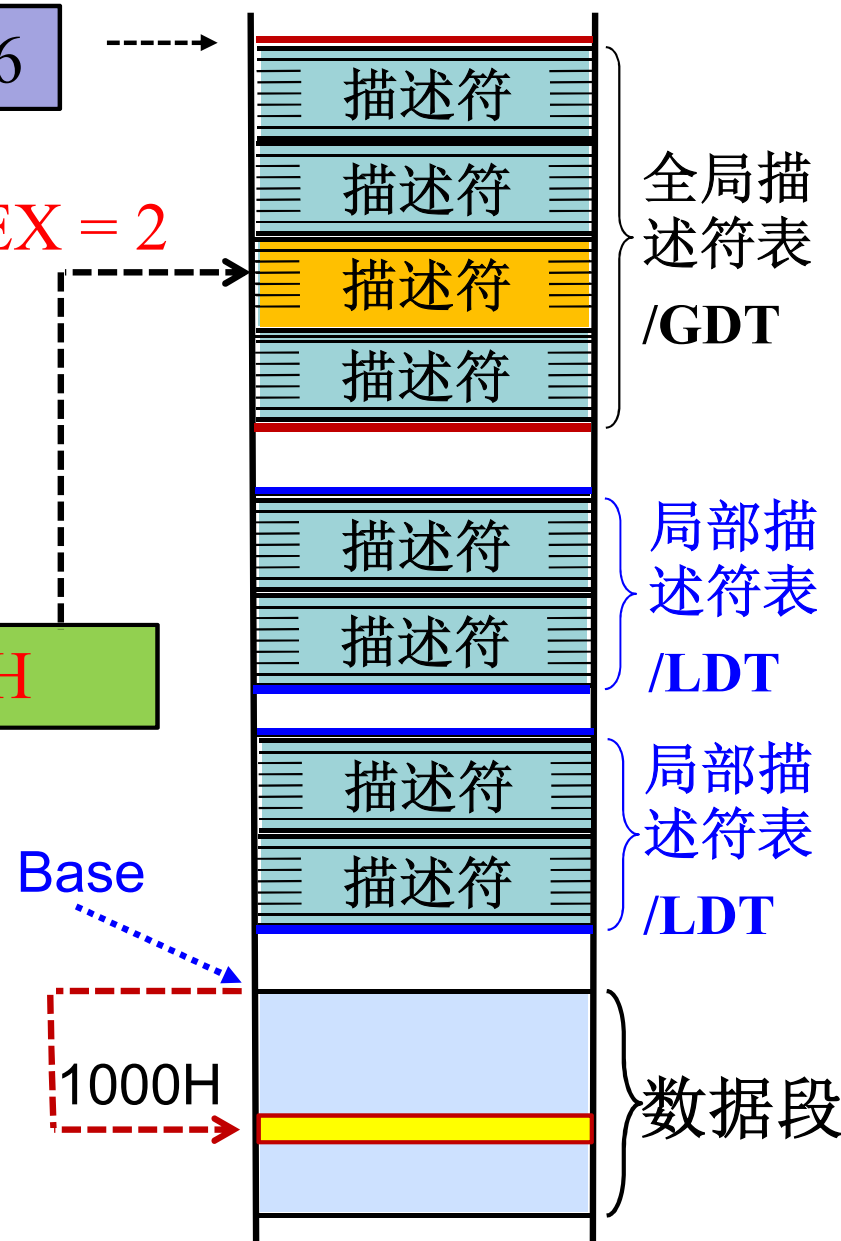
DS

13H



字节7	字节6	字节5	字节4	字节3	字节2	字节1	字节0
Base 2 (31...24)	Attribute		Base 1 (23...0)			Limit 1 (15...0)	

Base = Base 2 | Base1



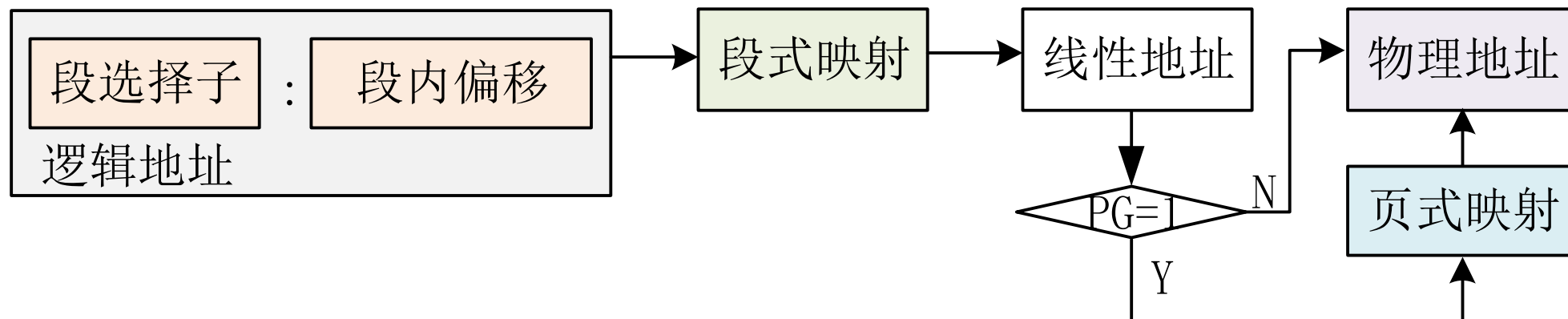
- 两种地址映射

- 段式地址映射

- 页式地址映射

```
MOV CR3, PT0_base ; 更换为页表0  
CALL LinearAddrDemo
```

```
MOV CR3, PT1_base ; 更换为页表1  
CALL LinearAddrDemo
```



- MMU: Memory Management Unit

● 权限级定义

■ **DPL** | RPL | CPL

字节7	字节6	字节5	字节4	字节3	字节2	字节1	字节0
Base 2 (31...24)	Attribute		Base 1 (23...0)			Limit 1 (15...0)	

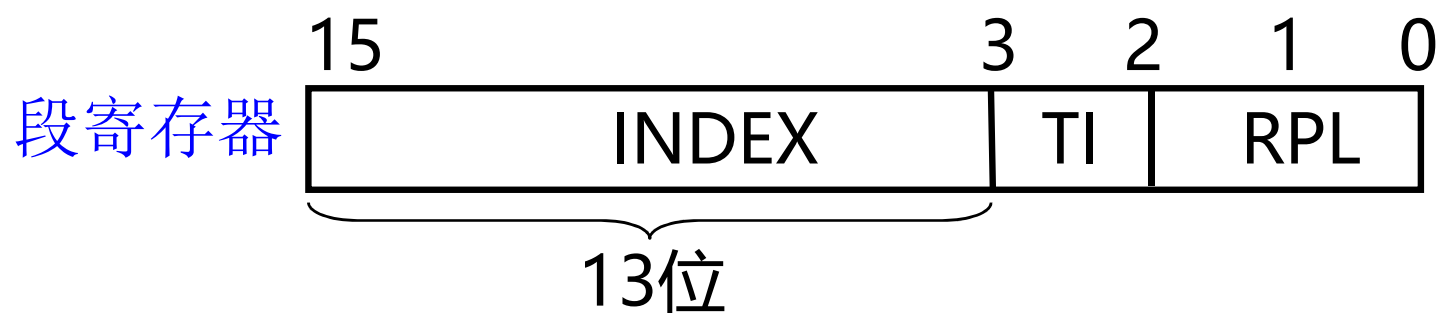
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
G	D	0	AVL	Limit 2 (19...16)				P	DPL		S	TYPE			

- 权限级定义

- DPL | RPL | CPL

- 请求特权级域

- Request Privilege Level

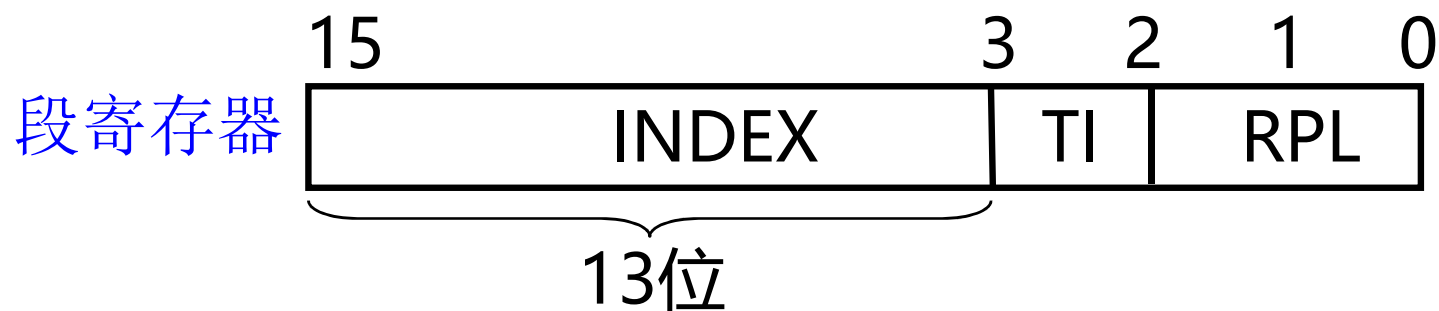


- 权限级定义

- DPL | RPL | **CPL**

- 当前特权级域 (CPL)

- Current Privilege Level



【回顾】任务状态段（Task State Segment）（104字节）

● 保存任务上下文和基本属性

（LDT Selector: LDT）

（CR3:PDBR, 页目录基址）

31	16	15	0	
I/O Map Base Address			T	100
			LDT Segment Selector	96
			GS	92
			FS	88
			DS	84
			SS	80
			CS	76
			ES	72
EDI				68
ESI				64
EBP				60
ESP				56
EBX				52
EDX				48
ECX				44
EAX				40
EFLAGS				36
EIP				32
CR3 (PDBR)				28
			SS2	24
ESP2				20
			SS1	16
ESP1				12
			SS0	8
ESP0				4
			Previous Task Link	0

【回顾】任务状态段（Task State Segment）- 高48字节

- 保存任务上下文和基本属性（LDT Selector: LDT）

31	16	15	0	
I/O Map Base Address			T	100
			LDT Segment Selector	96
			GS	92
			FS	88
			DS	84
			SS	80
			CS	76
			ES	72
EDI				68
ESI				64
EBP				60
ESP				56

【回顾】任务状态段（Task State Segment） - 高48字节

- 保存任务上下文和基本属性（CR3:PDBR, 页目录基址）

31	16 15	0	
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
		SS2	24
ESP2			20
		SS1	16
ESP1			12
		SS0	8
ESP0			4
		Previous Task Link	0

- 任务定义

- 代码，数据，堆栈，上下文

- LDT, TSS

- 任务的数据结构

- LDT（局部描述符表）

- ◆ LDT描述符(S=0)

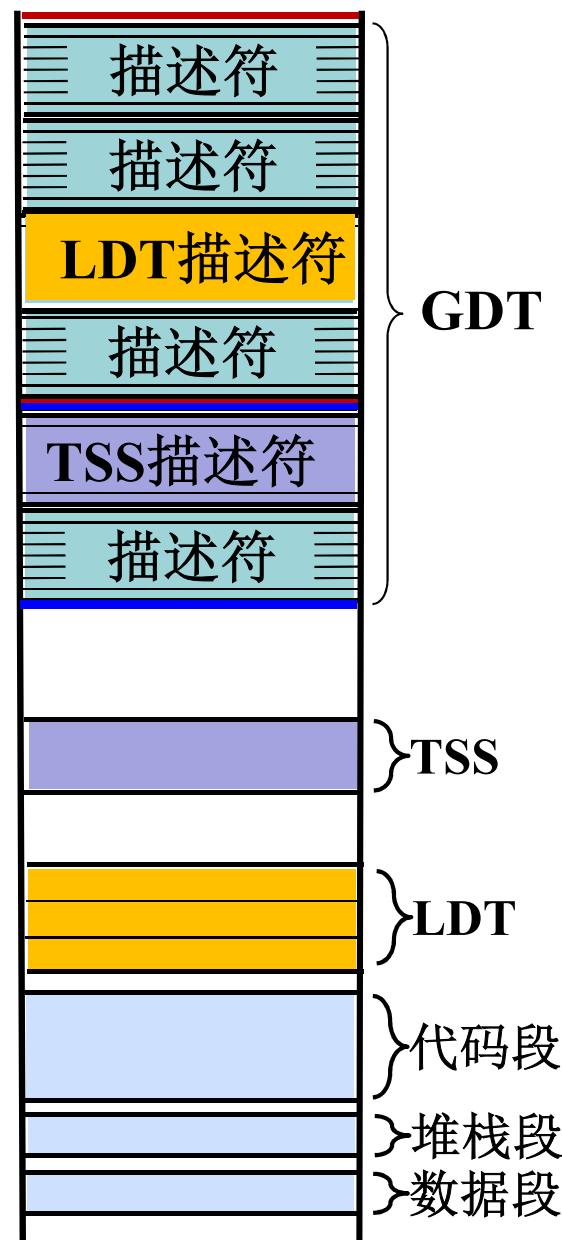
- ◆ 描述符(S=1)

- ◆ LDTR/lldtr指令

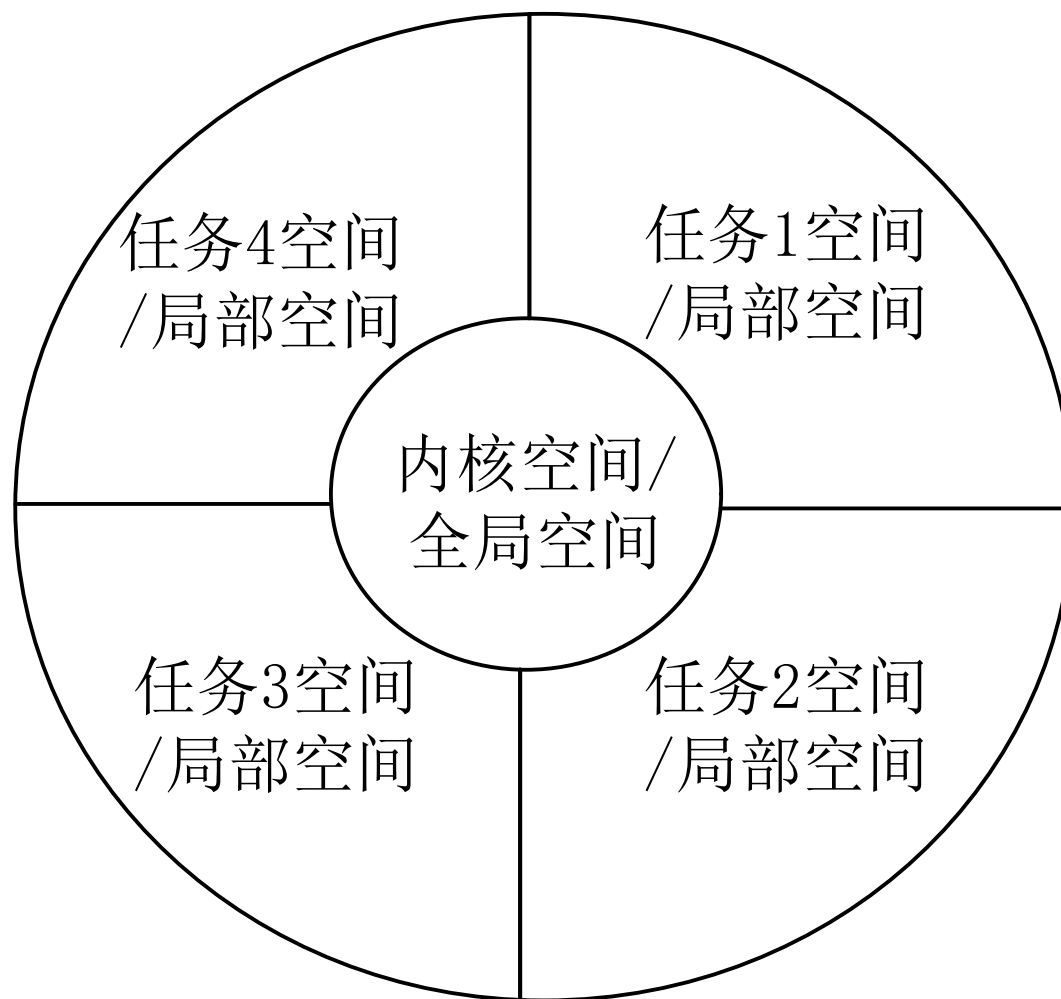
- TSS（任务状态段）

- ◆ TSS描述符(S=0)

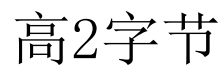
- ◆ TR/ltr指令



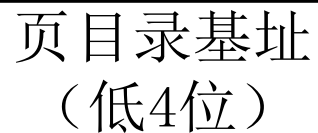
● 任务的隔离



■ 页目录基址：高20位



低2字节



例：建立两个任务

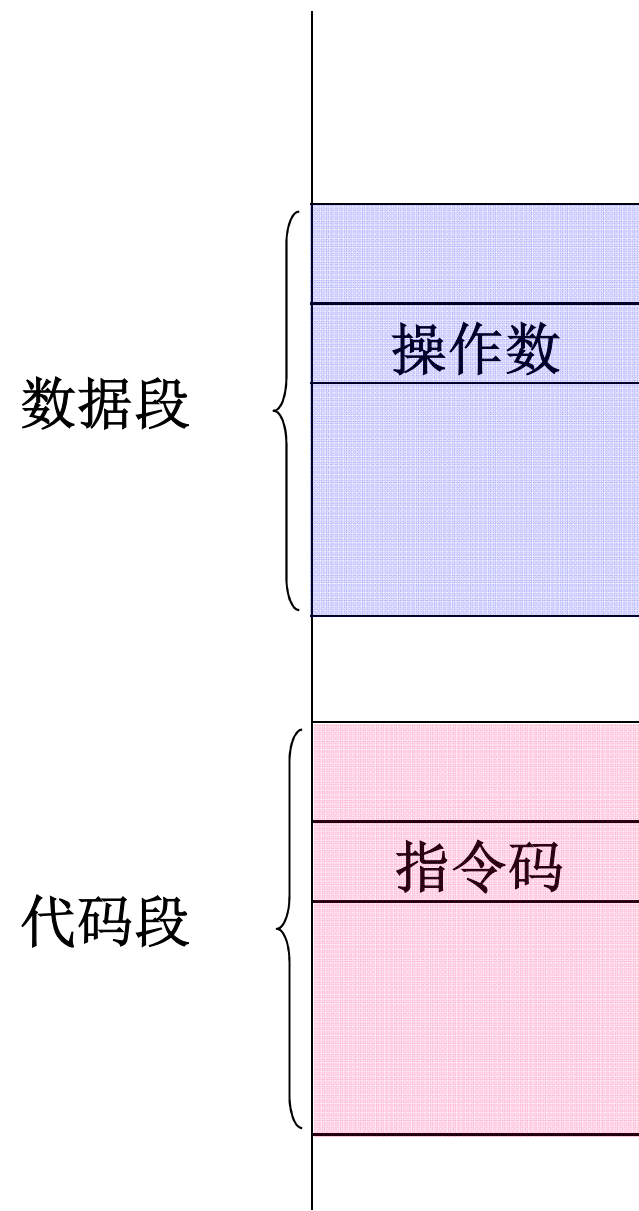
```
376 tss0:
377     dd 0
378     dd stack0_krn_ptr, 0x10 ;任务0的核心态使用的堆栈,就紧跟在 TSS0 的后面!
379     dd 0,0
380     dd 0,0
381     dd 0
382     dd task0                ;确保第一次切换到任务0时EIP从这里取值,故从task0处开始运行,
383     |                       ;因为任务0的基址为 0x10000,和核心一样,不指定这个的话,
384     |                       ;第一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
```

例：建立两个任务

```
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
```

回顾：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
 - ...



回顾：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			

回顾：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

回顾： 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
```

回顾: GDT的例子

```
22 [SECTION .gdt] ; LABEL_DESC_FLAT_C_3:, LABEL_DESC_FLAT_RW:
23 ; GDT
24 ;
25 LABEL_GDT: Descriptor 0, 0, 0
26 LABEL_DESC_NORMAL: Descriptor 0, 0ffffh, DA_DR
27 LABEL_DESC_FLAT_C_3:Descriptor 0, 0ffffffh, DA_CR
28 ;用户代码段
29 LABEL_DESC_FLAT_RW: Descriptor 0, 0ffffffh, DA_DR
30 ;系统数据段
31 LABEL_DESC_CODE32: Descriptor 0, SegCode32Len - 1, DA_CR
32 LABEL_DESC_CODE16: Descriptor 0, 0ffffh, DA_C
33 LABEL_DESC_DATA: Descriptor 0, DataLen - 1, DA_DR
34 LABEL_DESC_STACK: Descriptor 0, TopOfKStack, DA_D
35 LABEL_DESC_VIDEO_3: Descriptor 0B8000h, 0ffffh, DA_DR
36 LABEL_DESC_LDT0: Descriptor 0, LDT0Len-1, DA_LD
37 LABEL_DESC_TSS0: Descriptor 0, TSS0Len-1, DA_38
38 LABEL_DESC_LDT1: Descriptor 0, LDT1Len-1, DA_LD
39 LABEL_DESC_TSS1: Descriptor 0, TSS1Len-1, DA_38
40 ; GDT 结束
41 GdtLen EQU $ - LABEL_GDT ; GDT长度
42 GdtPtr DW GdtLen - 1 ; GDT界限
43 DD 0 ; GDT基地址
```

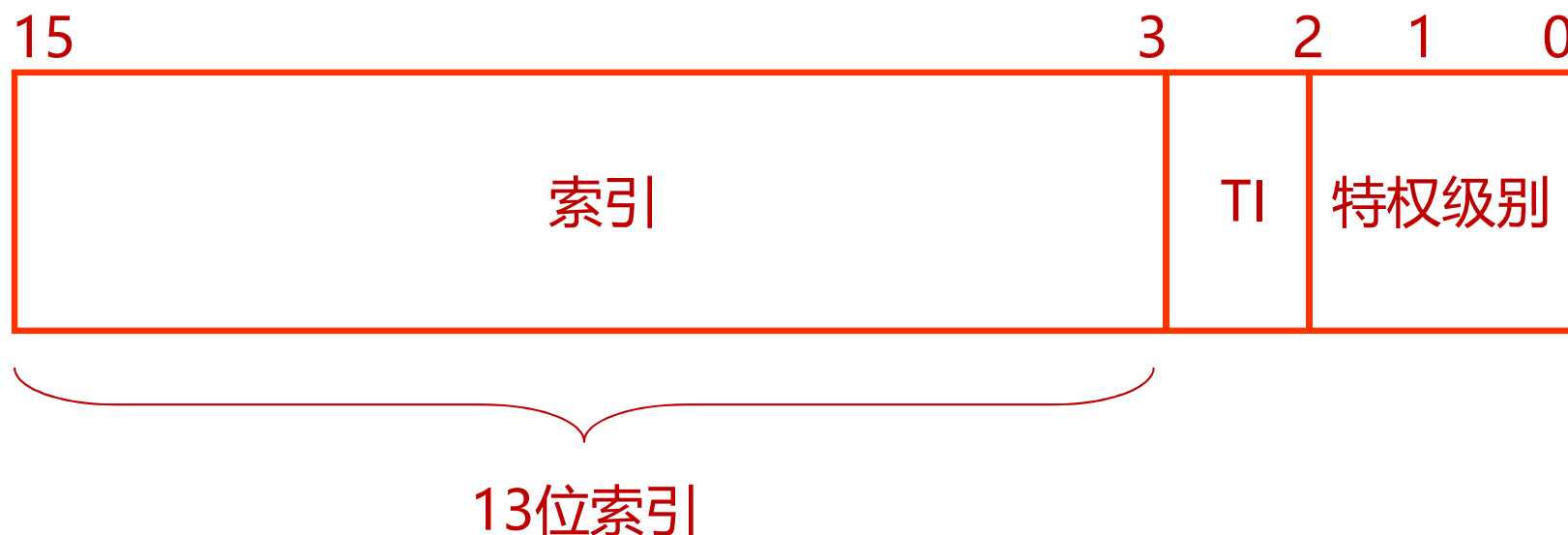
回顾：典型的段描述符

● 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

回顾：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



回顾：保护模式的含义：地址映射机制（段机制）

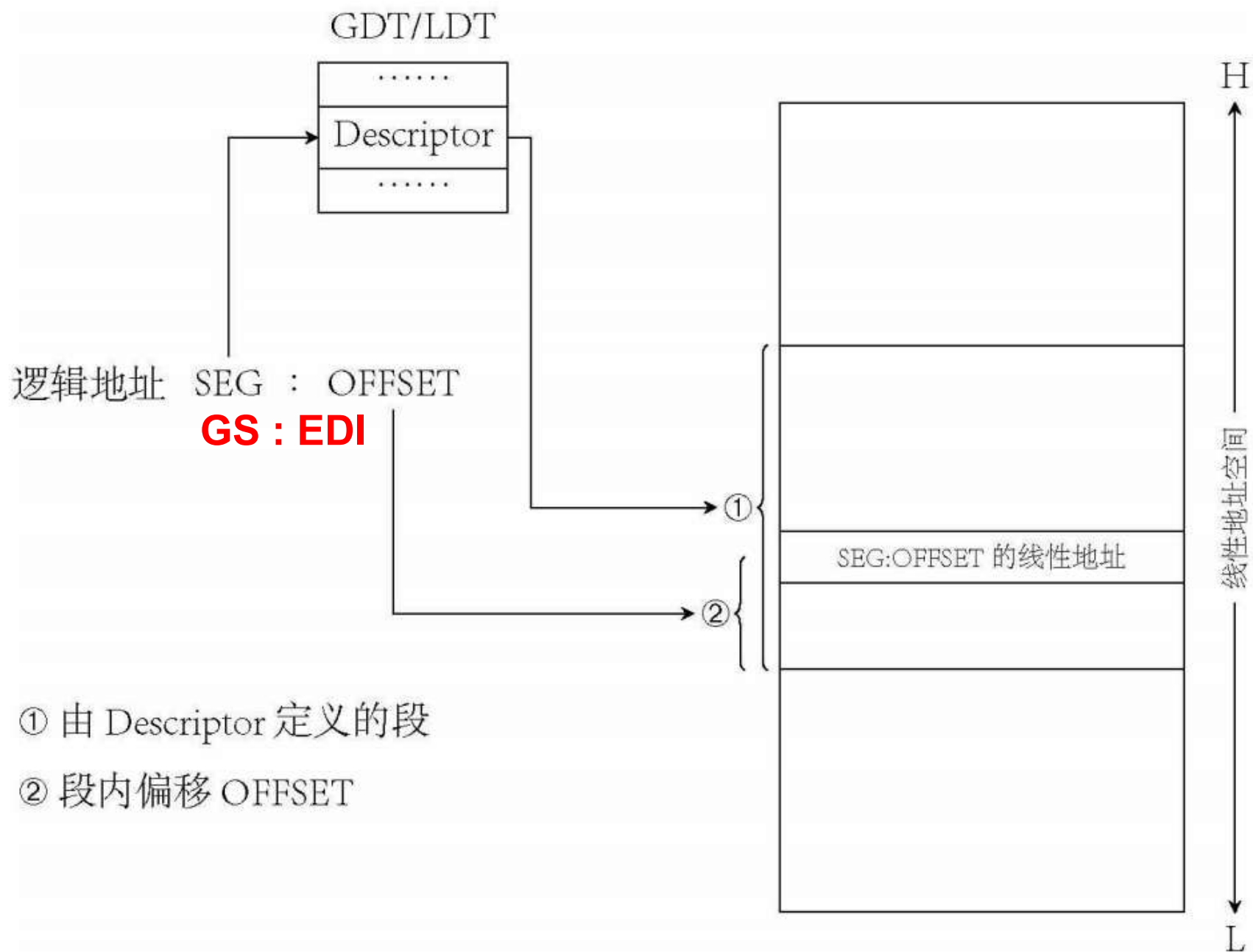


图3.6 段式寻址示意图

进入保护模式的步骤【参考《一个操作系统的实现》CH.3】

- 1. 创建一个有效的全局描述符
 - 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符
 - 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【参考《一个操作系统的实现》CH.3】


```
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
```


进入保护模式的步骤【参考《一个操作系统的实现》CH.3】

```
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
```

进入保护模式的步骤【参考《一个操作系统的实现》CH.3】

```
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
```



进入保护模式的步骤【参考《一个操作系统的实现》CH.3】

```
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
```

进入保护模式的步骤【参考《一个操作系统的实现》CH.3】

```
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
```



课设涉及的基本概念

- 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描述符存储到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

- 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

例：建立两个任务

```
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
```

例：建立两个任务

```
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
```

例：建立两个任务

```
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
```

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

- 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）【内容3】

```
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
V DWORD [currentTask]
```

● 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

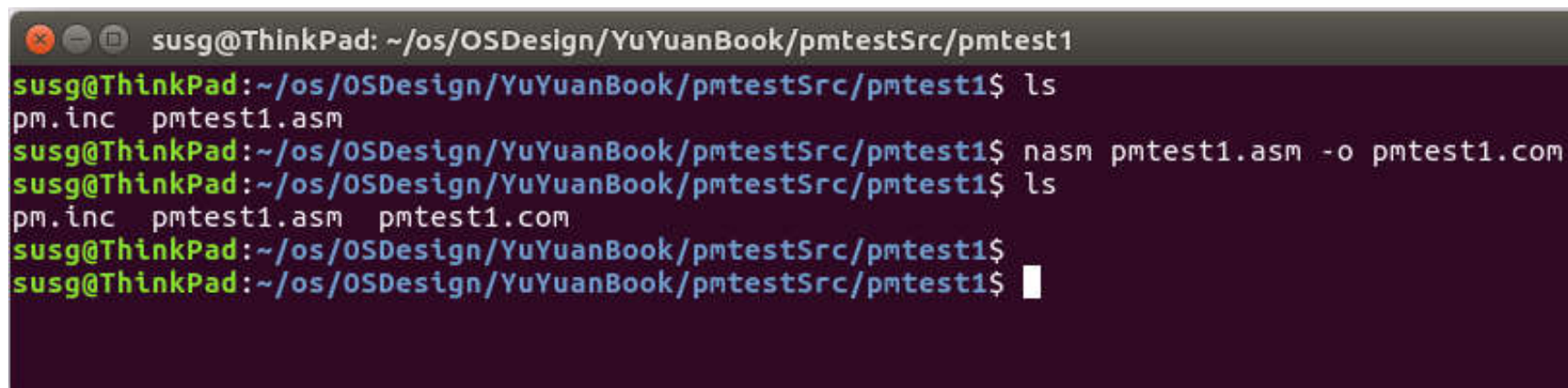
```
susg@ThinkPad: ~/os/OSDesign/YuYuanBook
susg@ThinkPad:~/os/OSDesign/YuYuanBook$ sudo apt-get install nasm
[sudo] password for susg:
Reading package lists... Done
Building dependency tree
Reading state information... Done
nasm is already the newest version (2.11.08-1ubuntu0.1).
0 upgraded, 0 newly 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

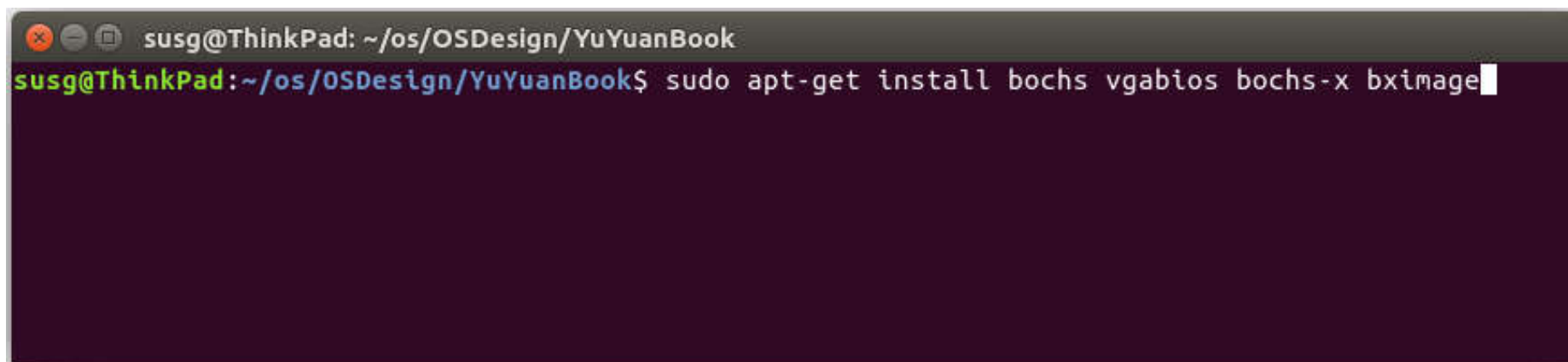
- `nasm pmtest1.asm -o pmtest1.com`



```
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

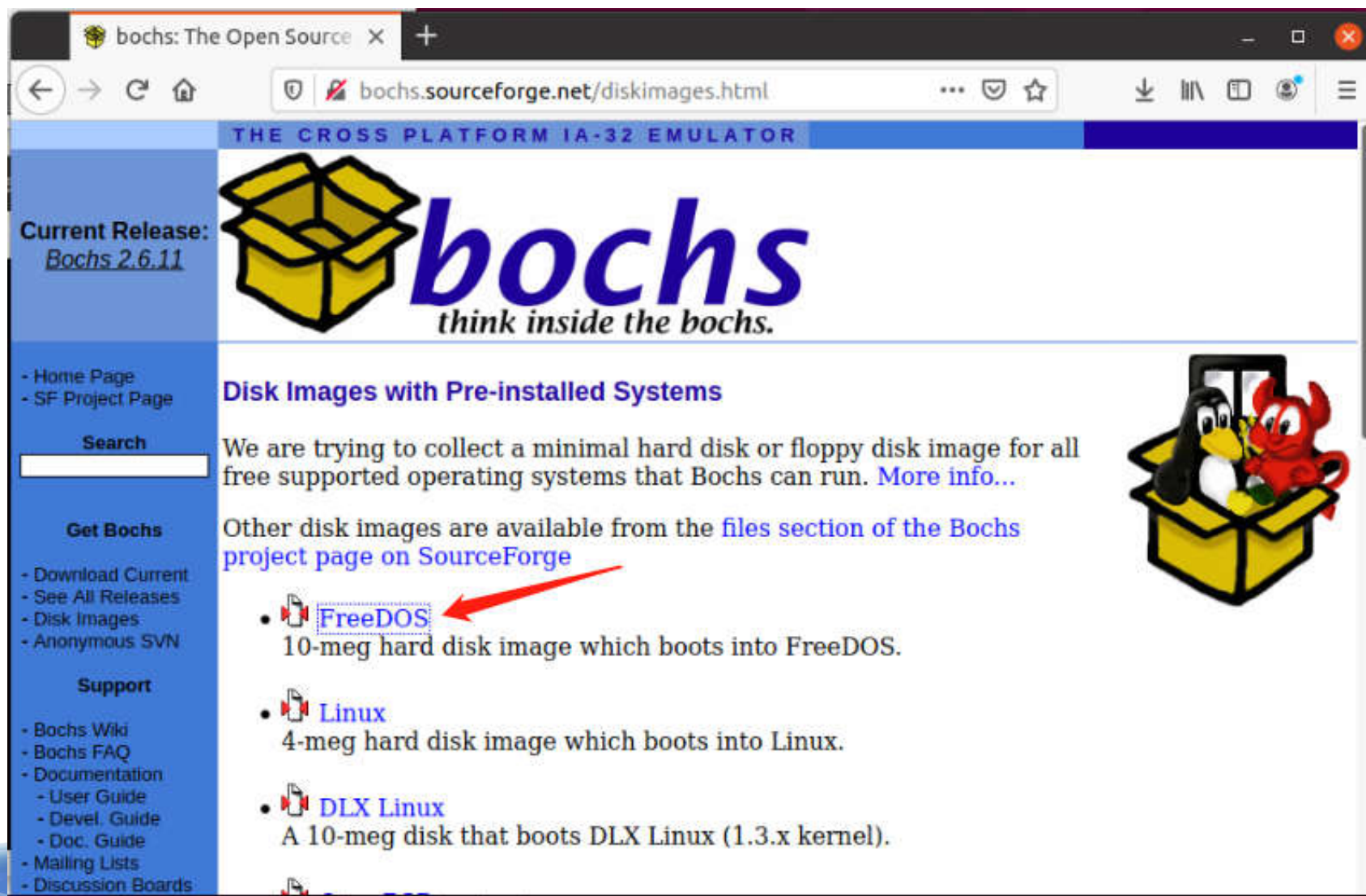
- 在bochs 官网下载freedos软盘映像文件



```
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bximage制作： pmtest.img

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

susg@ThinkPad:~/os/OSDesign/YuYuanBook$ bximage
=====
                        bximage
                Disk Image Creation Tool for Bochs
                $Id: bximage.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$
```

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 内存结构

■ 实际内存16M

任务0页目录和页表 →

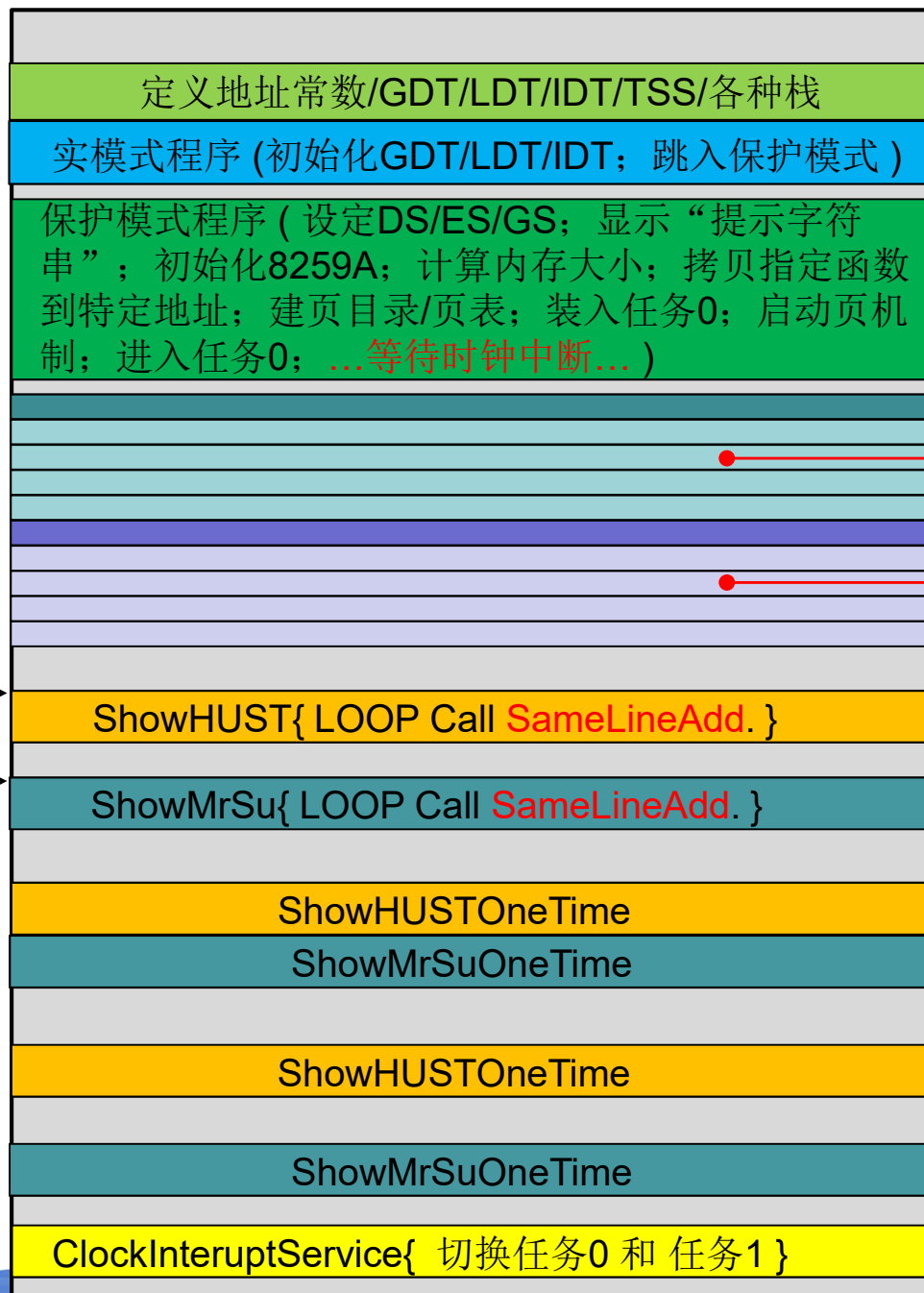
任务1页目录和页表 →

任务0的代码 →

任务1的代码 →

A0 →

A1 →



00 0000

SameLineAdd

SameLineAdd

FF FFFF

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 数据结构

■ GDT、段描述符、选择子、LDT、IDT、TSS等

```
22 [SECTION .gdt] ; LABEL_DESC_FLAT_C_3:, LABEL_DESC_FLAT_RW:
23 ; GDT
24 ;                                段基址,                段界限    , 属性
25 LABEL_GDT:                Descriptor    0,                0, 0
26 LABEL_DESC_NORMAL:        Descriptor    0,                0ffffh, DA_DR
27 LABEL_DESC_FLAT_C_3:Descriptor    0,                0ffffffh, DA_CR
28 LABEL_DESC_FLAT_RW:        Descriptor    0,                0ffffffh, DA_DR
29 LABEL_DESC_CODE32:        Descriptor    0,                SegCode32Len - 1, DA_CR
30 LABEL_DESC_CODE16:        Descriptor    0,                0ffffh, DA_C
31 LABEL_DESC_DATA:          Descriptor    0,                DataLen - 1, DA_DR
32 LABEL_DESC_STACK:         Descriptor    0,                TopOfKStack, DA_D
33 LABEL_DESC_VIDEO_3:        Descriptor    0B8000h,          0ffffh, DA_DR
34
35 LABEL_DESC_LDT0:           Descriptor    0,                LDT0Len-1, DA_LD
36 LABEL_DESC_TSS0:           Descriptor    0,                TSS0Len-1, DA_38
37 LABEL_DESC_LDT1:           Descriptor    0,                LDT1Len-1, DA_LD
38 LABEL_DESC_TSS1:           Descriptor    0,                TSS1Len-1, DA_38
39 ; GDT 结束
40
41 GdtLen      EQU $ - LABEL_GDT      ; GDT长度
42 GdtPtr      DW GdtLen - 1          ; GDT界限
43            DD 0                    ; GDT基地址
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 数据结构

■ GDT、段描述符、选择子、LDT、IDT、TSS等

```
61 [SECTION .ldt0]
62 ;
63 LABEL_LDT0: Descriptor 0,
64 LABEL_LDT0_DESC_CODE_3: Descriptor 0, Task0ShowHUSTCo
65 LABEL_LDT0_DESC_STACK3: Descriptor 0, TopOfTask0Stack
66 LABEL_LDT0_DESC_STACK0: Descriptor 0, TopOfTask0Stack
67
68 LDT0Len EQU $ - LABEL_LDT0 ; LDT长度
69 ; LDT0 选择子
70 SelectorLDT0Code_3 EQU (LABEL_LDT0_DESC_CODE_3 - LABEL
71 SelectorLDT0Stack3 EQU (LABEL_LDT0_DESC_STACK3 - LABEL
72 SelectorLDT0Stack0 EQU (LABEL_LDT0_DESC_STACK0 - LABEL
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 数据结构

■ GDT、段描述符、选择子、LDT、IDT、TSS等

```
485 LABEL_SEG_CODE32:      ;进入保护模式后的
486     MOV AX, SelectorData
487     MOV DS, AX           ; 数据段选择子
488     MOV ES, AX
489     MOV AX, SelectorVideo
        SpuriousHandler, SetupTask0Paging, F
490     MOV GS, AX           ; 视频段选择子
491
492     MOV AX, SelectorStack
493     MOV SS, AX           ; 堆栈段选择子
494     MOV ESP, TopOfKStack
495
496     CALL SetProtectMode8259A ; L551
497     ; 显示字符串:  szPMMessage
498     PUSH  szPMMessage
499     CALL  DispStr
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 数据结构

■ GDT、段描述符、选择子、LDT、IDT、TSS等

```
45 ; GDT 选择子
46 SelectorNormal EQU LABEL_DESC_NORMAL - LABEL_GDT
47 SelectorFlatC_3 EQU LABEL_DESC_FLAT_C_3 - LABEL_GDT + SA_RPL3
48 SelectorFlatRW EQU LABEL_DESC_FLAT_RW - LABEL_GDT
49 SelectorCode32 EQU LABEL_DESC_CODE32 - LABEL_GDT
50 SelectorCode16 EQU LABEL_DESC_CODE16 - LABEL_GDT
51 SelectorData EQU LABEL_DESC_DATA - LABEL_GDT
52 SelectorStack EQU LABEL_DESC_STACK - LABEL_GDT
53 SelectorVideo EQU LABEL_DESC_VIDEO_3 - LABEL_GDT
54
55 SelectorLDT0 EQU LABEL_DESC_LDT0 - LABEL_GDT
56 SelectorTSS0 EQU LABEL_DESC_TSS0 - LABEL_GDT
57 SelectorLDT1 EQU LABEL_DESC_LDT1 - LABEL_GDT
58 SelectorTSS1 EQU LABEL_DESC_TSS1 - LABEL_GDT
59 ; END of [SECTION .gdt]
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 数据结构

■ GDT、段描述符、选择子、LDT、IDT、TSS等

```
191 LABEL_IDT:
192 ;      门      目标选择子,      偏移,      DCount,      属性
193 %rep 32 ;重复填写32个无关紧要的中断
194      Gate SelectorCode32, SpuriousHandler,      0,      DA_386IGate
195 %endrep
196      ;20h是时钟中断
197 .020h: Gate SelectorCode32, ClockHandler,      0,      DA_386IGate
198 %rep 95 ;重复填写95个无关紧要的中断
199      Gate SelectorCode32, SpuriousHandler,      0,      DA_386IGate
200 %endrep
201      ;80h是系统调用
202 .080h: Gate SelectorCode32, UserIntHandler,      0,      DA_386IGate
203
204 IdtLen      EQU $ - LABEL_IDT
205 IdtPtr      DW      IdtLen - 1      ; 段界限
206            DD      0      ; 基地址
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 数据结构

■ GDT、段描述符、选择子、LDT、IDT、TSS等

```
629 _ClockHandler:
630 ClockHandler    EQU _ClockHandler - $$    ;参照当前节[
631
632     MOV EAX,1
633     CMP [currentTask],EAX ; 检查当前任务：是0还是1 ?
634
635     JE CUR_Task_IS_1
636
637     MOV [currentTask],EAX ;更新任务号 currentTask = 1
638     JMP SelectorTSS1:0    ; 切换到任务1 ; 偏移会被忽略
639     JMP EXIT_INT         ; 在调用上面一条指令进行切换时
640
641 CUR_Task_IS_1:    ; currentTask = 1
642
643     MOV DWORD [currentTask],0 ;更新任务号 currentTask
644     JMP SelectorTSS0:0    ;切换到任务0
645
646 EXIT_INT:
647
648     IRETD
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 特定代码段

■ 显示HUST一次 | 显示MrSU一次

```
841 ShowHUSTOneTime:
842 OffsetShowHUSTOneTime EQU ShowHUSTOneTime -
843     MOV AH, 0Ch                ; 0000: 黑底    1:
844     MOV AL, 'H'
845     MOV [GS:((80 * 17 + 0) * 2)], AX    ; 屏幕
846     MOV AL, 'U'
847     MOV [GS:((80 * 17 + 1) * 2)], AX    ; 屏幕
848     MOV AL, 'S'
849     MOV [GS:((80 * 17 + 2) * 2)], AX    ; 屏幕
850     MOV AL, 'T'
851     MOV [GS:((80 * 17 + 3) * 2)], AX    ; 屏幕
852     RETF                      ; RETF 实现段间返回, IP, CS 先后t
853 LenShowHUSTOneTime EQU $ - ShowHUSTOneTime

856 ShowMrSuOneTime:
857 OffsetShowMrSuOneTime EQU ShowMrSuOneTime -
858     MOV AH, 0Ch                ; 0000: 黑底    11
859     MOV AL, 'M'
860     MOV [GS:((80 * 17 + 0) * 2)], AX    ; 屏幕
861     MOV AL, 'r'
862     MOV [GS:((80 * 17 + 1) * 2)], AX    ; 屏幕
863     MOV AL, 'S'
864     MOV [GS:((80 * 17 + 2) * 2)], AX    ; 屏幕
865     MOV AL, 'u'
866     MOV [GS:((80 * 17 + 3) * 2)], AX    ; 屏幕
867     RETF                      ; RETF 实现段间返回, IP, CS 先后t
868 LenShowMrSuOneTime EQU $ - ShowMrSuOneTime
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 特定代码段

■ 进入保护模式的代码...

```
485 LABEL_SEG_CODE32:      ;进入保护模式后的第一段代码
486     MOV AX, SelectorData
487     MOV DS, AX           ; 数据段选择子
488     MOV ES, AX
489
490     CALL CopyFunToAssignedMem ; 完成两个任务
491                                     ; ~ L743 和
                                     ; ProcFoo 和 ProcBar
492
493     CALL InitTasksPageTables ; 演示改变页表
494     CALL LoadPDRTask0
495     CALL EnablePageMechanism
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 特定代码段

- 把“显示HUST一次”，“显示MrSU一次”拷贝到特定地址。

```
692 ; 拷贝两个用户函数到特定物理地址上--
693 CopyFunsToAssignedMem: ; L513调用:
694     MOV AX, CS
695     MOV DS, AX ; 为 L752和
696     MOV AX, SelectorFlatRW
697     MOV ES, AX ; 为 L752和
698
699     PUSH LenShowHUSTOneTime
700     PUSH OffsetShowHUSTOneTime
701     PUSH ProcShowHUSTOneTime
702     CALL MemCpy ; MemCpy(vo
703     ADD ESP, 12 ; ~ MemCpy
704
705     PUSH LenShowMrSuOneTime
706     PUSH OffsetShowMrSuOneTime
707     PUSH ProcShowMrSuOneTime
708     CALL MemCpy ; MemCpy(vo
709     ADD ESP, 12 ; ~ MemCpy
710     RET
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 特殊地址/数据结构

■ 特定代码段：显示HUST

```
8 PageDirBase0 EQU 200000h ; 页目录开始地址: 2M
9 PageTblBase0 EQU 201000h ; 页表开始地址: 2M + 64K
10 PageDirBase1 EQU 210000h ; 页目录开始地址: 2M + 64K + 4K
11 PageTblBase1 EQU 211000h ; 页表开始地址: 2M + 64K + 4K
12
13 LinearAddrDemo EQU 00401000h
14 ProcShowHUSTOneTime EQU 00401000h
15 ProcShowMrSuOneTime EQU 00501000h
```

代码缺行，伪代码，仅供思路参考。



程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 两个任务

```
923 [SECTION .task0]
924 [BITS 32]
925 ; ShowHUSTOneTime -----
926 LABEL_TASK0_ShowHUST_CODE3:
927
928     CALL SelectorFlatC_3 : LinearAddrDemo
929
930     MOV ECX,0xFFFFFFFF ;延时：延长字符串的显示时间
931 DELAY_A:
932     LOOP DELAY_A
933
934     JMP LABEL_TASK0_ShowHUST_CODE3
935
936 Task0ShowHUSTCode3Len EQU $ - LABEL_TASK0_ShowHUST_CODE3
937
939 [SECTION .task1]
940 [BITS 32]
941 ; ShowMrSuOneTime -----
942 LABEL_TASK1_ShowMrSu_CODE3:
943
944     CALL SelectorFlatC_3 : LinearAddrDemo
945
946     MOV ECX,0xFFFFFFFF ;延时：延长字符串的显示时间
947 DELAY_B:
948     LOOP DELAY_B
949
950     JMP LABEL_TASK1_ShowMrSu_CODE3
951 Task1ShowMrSuCode3Len EQU $ - LABEL_TASK1_ShowMrSu_CODE3
```

代码缺行，伪代码，语法不对，思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 两个任务

```
75 LABEL_TSS0:
76     DD 0
77     DD TopOfTask0Stack0,SelectorLDT0Stack0
78     DD 0,0
79     DD 0,0
80     DD PageDirBase0 ; CR3(PDBR)
81     DD 0
82     DD 0x200
83     DD 0,0,0,0 ;EAX,ECX,EDX,EBX
84     DD TopOfTask0Stack3,0,0,0
85     DD 0, SelectorLDT0Code_3, SelectorLDT0Stack3 ,(
86     DD SelectorLDT0 ;LDT
87     DD 0x08000000
88 TSS0Len EQU $ - LABEL_TSS0 ; TSS长度
89 LABEL_TASK0_STACK0:
90     TIMES 512 DB 0
91
92 TopOfTask0Stack0 EQU $ - LABEL_TASK0_STACK0 - 1
```

代码缺行，伪代码，仅供思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

```
788 InitTask1PageTable:
789     ; 根据内存大小计算应初始化页目
790     MOV [PageTableNumber], ECX ;
791     ; 初始化页目录
792     MOV AX, SelectorFlatRW ;因为
793     MOV ES, AX ;
794     MOV EDI, PageDirBase1 ; 此段
795     XOR EAX, EAX
796     MOV EAX, PageTblBase1 | PG_P
797     MOV ECX, [PageTableNumber]
798 .NEXT_PDE:
799     STOSD
800     ADD EAX, 4096 ; 为了
801     LOOP .NEXT_PDE
802
803     ; 再初始化所有页表
804     MOV EAX, [PageTableNumber] ;
805     MOV EBX, 1024 ; 每个
806     MUL EBX
807     MOV ECX, EAX ; PTE
808     MOV EDI, PageTblBase1 ; 此段
809     XOR EAX, EAX
810     MOV EAX, PG_P | PG_USU | PG_F
811 .NEXT_PTE:
812     STOSD
813     ADD EAX, 4096 ; 每一
814     LOOP .NEXT_PTE
815
816     RET
```

```
787 ; 初始化任务1的页目录和页表 -----
788 InitTask1PageTable:
789     ; 根据内存大小计算应初始化页目录多少个PD
790     .....
791 .NEXT_PTE:
792     STOSD
793     ADD EAX, 4096 ; 每一页指向 4K
794     LOOP .NEXT_PTE
795
796     ; 在此假设内存是大于 8M 的
797     MOV EAX, LinearAddrDemo ;修改线性地址 L
798     SHR EAX, 22 ;获得最高10位 (即
799     MOV EBX, 4096
800     MUL EBX ;4K × 最高10位 (
801     MOV ECX, EAX ;ECX = 对应页表
802     MOV EAX, LinearAddrDemo
803     SHR EAX, 12 ;获得最高20位 (即
804     AND EAX, 03FFh ;最高20位 位与
805     MOV EBX, 4 ;每个PTE占4个字
806     MUL EBX ;计算该PTE在页表
807     ADD EAX, ECX ;计算该PTE的偏移
808     ADD EAX, PageTblBase1 ;EAX = PTE在数
809     MOV DWORD [ES:EAX], ProcShowMrSuOneTime
810
811     RET
```

代码缺行，伪代码，仅供思路参考。

程序主体结构（级别3：两个任务在时钟驱动下交替切换）

● 特定代码段

■ 进入保护模式的代码...

```
485 LABEL_SEG_CODE32:           ;进入保护模式后的第一段代码：页
486
487     .....
488     MOV EAX,SelectorTSS0
489     LTR AX
490     MOV EAX,SelectorLDT0
491     LLDT AX                   ;加载局部描述符表寄存器
492     MOV DWORD [currentTask],0
493     STI                       ;开中断,此时实时时钟的中
494
495     PUSH SelectorLDT0Stack3
496     PUSH TopOfTask0Stack3
497     PUSHF                     ;标志寄存器psw入栈
498     PUSH SelectorLDT0Code_3
499     PUSH 0
500     IRET                     ; RETF ;若改为RETF是不能进入任务0的。
501
502     JMP $                     ; 跳到当前地址，死循环
```

代码缺行，伪代码，仅供思路参考。