```
; ==========================================
; superv2.asm
; 编译方法：nasm superv2.asm -o superv2.com
; 实现功能：   支持简易PCB（名字，pid，静态优先级，动态优先级），即支持多个简易进程，LDT，TSS，页目录，页表无需手动编写。
;                       使用段页式地址映射，之前一版共用页目录与页表。
;                       此版本为每个简易进程创建页目录和页表，但为了简化，页目录和页表均直接将线性地址直接映射为物理地址。
; ==========================================

%include        "pm.inc"        ; 常量, 宏, 以及一些说明

%define SetNumberOfProcess 5

%assign                 PhysicalAddress         200000h
%assign                 num                                     1
%rep                    SetNumberOfProcess
PageDirBase%[num]       equ     PhysicalAddress ; 页目录开始地址:       2M
%assign                 PhysicalAddress         PhysicalAddress + 01000H
PageTblBase%[num]       equ     PhysicalAddress ; 页表开始地址:         2M +  4K
%assign                 PhysicalAddress         PhysicalAddress + 0F000H
%assign                 num                                     num + 1
%endrep

ProcPagingDemo          equ     301000h
LinearAddrDemo          equ     401000h

%assign                 PhysicalAddress         401000h
%assign                 num                                     1
%rep                    SetNumberOfProcess
ProcTask%[num]          equ     PhysicalAddress
%assign                 num                                     num + 1
%assign                 PhysicalAddress         PhysicalAddress + 100000H
%endrep

org     0100h
jmp     LABEL_BEGIN

[SECTION .gdt]
; GDT
;                                       段基址,       段界限     , 属性
LABEL_GDT:                      Descriptor              0,                  0, 0                            ; 空描述符
LABEL_DESC_NORMAL:      Descriptor              0,              0ffffh, DA_DRW                  ; Normal 描述符
```

```
42 LABEL_DESC_FLAT_C:      Descriptor         0,           0fffffh, DA_CR | DA_32 | DA_LIMIT_4K; 0 ~ 4G
43 LABEL_DESC_FLAT_RW:     Descriptor         0,           0fffffh, DA_DRW | DA_LIMIT_4K   ; 0 ~ 4G
44 LABEL_DESC_CODE32:      Descriptor             0,  SegCode32Len - 1, DA_CR | DA_32      ; 非一致代码段, 32
45 LABEL_DESC_CODE16:      Descriptor             0,            0ffffh, DA_C                       ; 非一致代码段, 16
46 LABEL_DESC_DATA:        Descriptor             0,               DataLen - 1, DA_DRW                                ; Data
47 LABEL_DESC_STACK:       Descriptor             0,        TopOfStack, DA_DRWA | DA_32; Stack, 32 位
48
49 %assign                 num     1
50 %rep                    SetNumberOfProcess
51 LABEL_DESC_STACK%[num]: Descriptor         0,  TopOfStack%[num], DA_DRWA | DA_32; Stack, 32 位
52 LABEL_DESC_LDT%[num]:   Descriptor         0,  LDT%[num]Len - 1, DA_LDT                ; LDT
53 LABEL_DESC_TSS%[num]:   Descriptor         0,  TSSLen%[num] - 1, DA_386TSS             ; TSSs
54 %assign                 num     num + 1
55 %endrep
56
57 LABEL_DESC_VIDEO:       Descriptor       0B8000h,            0ffffh, DA_DRW                     ; 显存首地址
58 ; GDT 结束
59
60 GdtLen          equ     $ - LABEL_GDT   ; GDT长度
61 GdtPtr          dw      GdtLen - 1              ; GDT界限
62                  dd      0                               ; GDT基地址
63
64 ; GDT 选择子
65 SelectorNormal          equ     LABEL_DESC_NORMAL       - LABEL_GDT
66 SelectorFlatC           equ     LABEL_DESC_FLAT_C       - LABEL_GDT
67 SelectorFlatRW          equ     LABEL_DESC_FLAT_RW      - LABEL_GDT
68 SelectorCode32          equ     LABEL_DESC_CODE32       - LABEL_GDT
69 SelectorCode16          equ     LABEL_DESC_CODE16       - LABEL_GDT
70 SelectorData            equ     LABEL_DESC_DATA         - LABEL_GDT
71 SelectorStack           equ     LABEL_DESC_STACK        - LABEL_GDT
72
73 %assign                 num     1
74 %rep                    SetNumberOfProcess
75 SelectorStack%[num]             equ     LABEL_DESC_STACK%[num]          - LABEL_GDT
76 SelectorLDT%[num]               equ LABEL_DESC_LDT%[num]                - LABEL_GDT
77 SelectorTSS%[num]               equ LABEL_DESC_TSS%[num]                - LABEL_GDT
78 %assign                 num     num + 1
79 %endrep
80
81 SelectorVideo           equ     LABEL_DESC_VIDEO        - LABEL_GDT
82 ; END of [SECTION .gdt]
83
```

```
%macro PCBlock 4
        db      %1                              ;进程名，不能超过10个字符
        %strlen charcnt %1
        %rep    10 - charcnt
        db      0
        %endrep
        dw      %2                              ;pid
        dw      %3                              ;当前进程持有的时间片
        dw      %4                              ;静态优先级
%endmacro ; 共 16 字节

[SECTION .data1]         ; 数据段
ALIGN   32
[BITS   32]
LABEL_DATA:
; 实模式下使用这些符号
; 字符串
_szPMMessage:           db      "In Protect Mode now. ^-^", 0Ah, 0Ah, 0 ; 进入保护模式后显示此字符串
_szMemChkTitle:         db      "BaseAddrL BaseAddrH LengthLow LengthHigh   Type", 0Ah, 0       ; 进入保护模式后显示此字符串
_szRAMSize:                     db      "RAM size:", 0
_szReturn:                      db      0Ah, 0
; 变量
_wSPValueInRealMode:dw  0
_dwMCRNumber:           dd      0       ; Memory Check Result
_dwDispPos:                     dd      (80 * 6 + 0) * 2        ; 屏幕第 6 行, 第 0 列。
_dwMemSize:                     dd      0
_ARDStruct:                     ; Address Range Descriptor Structure
_dwBaseAddrLow:         dd      0
_dwBaseAddrHigh:        dd      0
_dwLengthLow:           dd      0
_dwLengthHigh:          dd      0
_dwType:                        dd      0
_PageTableNumber:       dd      0
_SavedIDTR:                     dd      0       ; 用于保存 IDTR
                                dd      0
_SavedIMREG:            db      0       ; 中断屏蔽寄存器值
_MemChkBuf:     times   256     db      0
_NumberOfProcess:       dd      SetNumberOfProcess
_nowProcess:            dd      0
_PCB:                           PCBlock "VERY.exe",             1,              0,              010H
                                        PCBlock "LOVE.exe",             2,              0,              0AH
                                        PCBlock "HUST.exe",             3,              0,              08H
```

```
                                    PCBlock "MRSU.exe",             4,              0,              06H
                                    PCBlock "1010.exe",             4,              0,              012H
_TSSArray:
%assign                 num     1
%rep                    SetNumberOfProcess
        dd                              0
        dw                              SelectorTSS%[num]
        dw                              0
%assign                 num     num + 1
%endrep



; 保护模式下使用这些符号
szPMMessage             equ     _szPMMessage    - $$
szMemChkTitle   equ     _szMemChkTitle  - $$
szRAMSize               equ     _szRAMSize      - $$
szReturn                equ     _szReturn       - $$
dwDispPos               equ     _dwDispPos      - $$
dwMemSize               equ     _dwMemSize      - $$
dwMCRNumber             equ     _dwMCRNumber    - $$
ARDStruct               equ     _ARDStruct      - $$
dwBaseAddrLow   equ     _dwBaseAddrLow  - $$
dwBaseAddrHigh  equ     _dwBaseAddrHigh - $$
dwLengthLow             equ     _dwLengthLow    - $$
dwLengthHigh    equ     _dwLengthHigh   - $$
dwType                  equ     _dwType         - $$
MemChkBuf               equ     _MemChkBuf      - $$
PageTableNumber equ     _PageTableNumber- $$
SavedIDTR               equ     _SavedIDTR      - $$
SavedIMREG              equ     _SavedIMREG     - $$
NumberOfProcess equ _NumberOfProcess - $$
nowProcess              equ _nowProcess - $$
PCB                     equ _PCB - $$
TSSArray                equ _TSSArray - $$
DataLen                 equ     $ - LABEL_DATA
; END of [SECTION .data1]


; 全局堆栈段
[SECTION .gs]
ALIGN   32
```

```
[BITS   32]
LABEL_STACK:
        times 512 db 0
TopOfStack      equ     $ - LABEL_STACK - 1
; END of [SECTION .gs]

%assign                 num     1
%rep                    SetNumberOfProcess
[SECTION .gs%[num]]
ALIGN   32
[BITS   32]
LABEL_STACK%[num]:
        times 512 db 0
TopOfStack%[num]        equ     $ - LABEL_STACK%[num] - 1
; END of [SECTION .gs%[num]]
%assign                 num     num + 1
%endrep

; 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]

[SECTION .s16]
[BITS   16]
LABEL_BEGIN:
        mov     ax, cs
```

```
        mov     ds, ax
        mov     es, ax
        mov     ss, ax
        mov     sp, 0100h

        mov     [LABEL_GO_BACK_TO_REAL+3], ax
        mov     [_wSPValueInRealMode], sp

        ; 得到内存数
        mov     ebx, 0
        mov     di, _MemChkBuf
.loop:
        mov     eax, 0E820h
        mov     ecx, 20
        mov     edx, 0534D4150h
        int     15h
        jc      LABEL_MEM_CHK_FAIL
        add     di, 20
        inc     dword [_dwMCRNumber]
        cmp     ebx, 0
        jne     .loop
        jmp     LABEL_MEM_CHK_OK
LABEL_MEM_CHK_FAIL:
        mov     dword [_dwMCRNumber], 0
LABEL_MEM_CHK_OK:

        ; 初始化 16 位代码段描述符
        mov     ax, cs
        movzx   eax, ax
        shl     eax, 4
        add     eax, LABEL_SEG_CODE16
        mov     word [LABEL_DESC_CODE16 + 2], ax
        shr     eax, 16
        mov     byte [LABEL_DESC_CODE16 + 4], al
        mov     byte [LABEL_DESC_CODE16 + 7], ah

        ; 初始化 32 位代码段描述符
        xor     eax, eax
        mov     ax, cs
        shl     eax, 4
        add     eax, LABEL_SEG_CODE32
        mov     word [LABEL_DESC_CODE32 + 2], ax
```

```
        shr     eax, 16
        mov     byte [LABEL_DESC_CODE32 + 4], al
        mov     byte [LABEL_DESC_CODE32 + 7], ah

        ; 初始化数据段描述符
        xor     eax, eax
        mov     ax, ds
        shl     eax, 4
        add     eax, LABEL_DATA
        mov     word [LABEL_DESC_DATA + 2], ax
        shr     eax, 16
        mov     byte [LABEL_DESC_DATA + 4], al
        mov     byte [LABEL_DESC_DATA + 7], ah

        ; 初始化堆栈段描述符
        xor     eax, eax
        mov     ax, ds
        shl     eax, 4
        add     eax, LABEL_STACK
        mov     word [LABEL_DESC_STACK + 2], ax
        shr     eax, 16
        mov     byte [LABEL_DESC_STACK + 4], al
        mov     byte [LABEL_DESC_STACK + 7], ah

        ; 初始化每一个进程
%assign                 num     1
%rep                    SetNumberOfProcess
        ; 初始化内核堆栈段描述符
        xor     eax, eax
        mov     ax, ds
        shl     eax, 4
        add     eax, LABEL_STACK%[num]
        mov     word [LABEL_DESC_STACK%[num] + 2], ax
        shr     eax, 16
        mov     byte [LABEL_DESC_STACK%[num] + 4], al
        mov     byte [LABEL_DESC_STACK%[num] + 7], ah

        ; 初始化 LDT 在 GDT 中的描述符
        xor     eax, eax
        mov     ax, ds
        shl     eax, 4
        add     eax, LABEL_LDT%[num]
```

```
294         mov     word [LABEL_DESC_LDT%[num] + 2], ax
295         shr     eax, 16
296         mov     byte [LABEL_DESC_LDT%[num] + 4], al
297         mov     byte [LABEL_DESC_LDT%[num] + 7], ah
298
299         ; 初始化 TSS 描述符
300         xor     eax, eax
301         mov     ax, ds
302         shl     eax, 4
303         add     eax, LABEL_TSS%[num]
304         mov     word [LABEL_DESC_TSS%[num] + 2], ax
305         shr     eax, 16
306         mov     byte [LABEL_DESC_TSS%[num] + 4], al
307         mov     byte [LABEL_DESC_TSS%[num] + 7], ah
308
309         ; 初始化 LDT 中的描述符
310         xor     eax, eax
311         mov     ax, ds
312         shl     eax, 4
313         add     eax, LABEL_Task%[num]
314         mov     word [LABEL_LDT%[num]_DESC_TASK + 2], ax
315         shr     eax, 16
316         mov     byte [LABEL_LDT%[num]_DESC_TASK + 4], al
317         mov     byte [LABEL_LDT%[num]_DESC_TASK + 7], ah
318
319         xor     eax, eax
320         mov     ax, ds
321         shl     eax, 4
322         add     eax, LABEL_Data%[num]
323         mov     word [LABEL_LDT%[num]_DESC_DATA + 2], ax
324         shr     eax, 16
325         mov     byte [LABEL_LDT%[num]_DESC_DATA + 4], al
326         mov     byte [LABEL_LDT%[num]_DESC_DATA + 7], ah
327
328         xor     eax, eax
329         mov     ax, ds
330         shl     eax, 4
331         add     eax, Local_LABEL_STACK%[num]
332         mov     word [LABEL_LDT%[num]_DESC_STACK + 2], ax
333         shr     eax, 16
334         mov     byte [LABEL_LDT%[num]_DESC_STACK + 4], al
335         mov     byte [LABEL_LDT%[num]_DESC_STACK + 7], ah
```

```
%assign                 num     num + 1
%endrep

        ; 为加载 GDTR 作准备
        xor     eax, eax
        mov     ax, ds
        shl     eax, 4
        add     eax, LABEL_GDT          ; eax <- gdt 基地址
        mov     dword [GdtPtr + 2], eax ; [GdtPtr + 2] <- gdt 基地址

        ; 为加载 IDTR 作准备
        xor     eax, eax
        mov     ax, ds
        shl     eax, 4
        add     eax, LABEL_IDT          ; eax <- idt 基地址
        mov     dword [IdtPtr + 2], eax ; [IdtPtr + 2] <- idt 基地址

        ; 保存 IDTR
        sidt    [_SavedIDTR]

        ; 保存中断屏蔽寄存器(IMREG)值
        in      al, 21h
        mov     [_SavedIMREG], al

        ; 加载 GDTR
        lgdt    [GdtPtr]

        ; 加载 IDTR
        lidt    [IdtPtr]

        ; 打开地址线A20
        in      al, 92h
        or      al, 00000010b
        out     92h, al

        ; 准备切换到保护模式
        mov     eax, cr0
        or      eax, 1
        mov     cr0, eax

        ; 真正进入保护模式
        jmp     dword SelectorCode32:0  ; 执行这一句会把 SelectorCode32 装入 cs, 并跳转到 Code32Selector:0  处
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

LABEL_REAL_ENTRY:               ; 从保护模式跳回到实模式就到了这里
        mov     ax, cs
        mov     ds, ax
        mov     es, ax
        mov     ss, ax
        mov     sp, [_wSPValueInRealMode]

        lidt    [_SavedIDTR]    ; 恢复 IDTR 的原值

        mov     al, [_SavedIMREG]       ; ┓恢复中断屏蔽寄存器(IMREG)的原值
        out     21h, al                         ; ┛

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

        sti                     ; 开中断

        mov     ax, 4c00h       ; ┓
        int     21h                     ; ┛回到 DOS
; END of [SECTION .s16]


[SECTION .s32]; 32 位代码段. 由实模式跳入.
[BITS   32]

LABEL_SEG_CODE32:
        mov     ax, SelectorData
        mov     ds, ax                  ; 数据段选择子
        mov     es, ax
        mov     ax, SelectorVideo
        mov     gs, ax                  ; 视频段选择子

        mov     ax, SelectorStack
        mov     ss, ax                  ; 堆栈段选择子

        mov     esp, TopOfStack

        call    Init8259A
```

```
420
421         ; 下面显示一个字符串
422         push    szPMMessage
423         call    DispStr
424         add     esp, 4
425
426         push    szMemChkTitle
427         call    DispStr
428         add     esp, 4
429
430         call    DispMemSize             ; 显示内存信息
431
432         call    StartPage               ; 开启分页机制
433
434
435         mov     ax, SelectorLDT1
436         lldt    ax
437         jmp     SelectorLDT1Task:0
438
439         call    SetRealmode8259A
440
441         ; 到此停止
442         jmp     SelectorCode16:0
443
444
445 ; Init8259A ---------------------------------------------------------------------------------------
446 Init8259A:
447         mov     al, 011h
448         out     020h, al        ; 主8259, ICW1.
449         call    io_delay
450
451         out     0A0h, al        ; 从8259, ICW1.
452         call    io_delay
453
454         mov     al, 020h        ; IRQ0 对应中断向量 0x20
455         out     021h, al        ; 主8259, ICW2.
456         call    io_delay
457
458         mov     al, 028h        ; IRQ8 对应中断向量 0x28
459         out     0A1h, al        ; 从8259, ICW2.
460         call    io_delay
461
```

```
462         mov     al, 004h        ; IR2 对应从8259
463         out     021h, al        ; 主8259, ICW3.
464         call    io_delay
465
466         mov     al, 002h        ; 对应主8259的 IR2
467         out     0A1h, al        ; 从8259, ICW3.
468         call    io_delay
469
470         mov     al, 001h
471         out     021h, al        ; 主8259, ICW4.
472         call    io_delay
473
474         out     0A1h, al        ; 从8259, ICW4.
475         call    io_delay
476
477         mov     al, 11111110b   ; 仅仅开启定时器中断
478         out     021h, al        ; 主8259, OCW1.
479         call    io_delay
480
481         mov     al, 11111111b   ; 屏蔽从8259所有中断
482         out     0A1h, al        ; 从8259, OCW1.
483         call    io_delay
484
485         ret
486 ; Init8259A ---------------------------------------------------------------------------------------------
487
488
489 ; SetRealmode8259A ---------------------------------------------------------------------------------------------
490 SetRealmode8259A:
491         mov     ax, SelectorData
492         mov     fs, ax
493
494         mov     al, 017h
495         out     020h, al        ; 主8259, ICW1.
496         call    io_delay
497
498         mov     al, 008h        ; IRQ0 对应中断向量 0x8
499         out     021h, al        ; 主8259, ICW2.
500         call    io_delay
501
502         mov     al, 001h
503         out     021h, al        ; 主8259, ICW4.
```

```
        call    io_delay

        mov     al, [fs:SavedIMREG]     ; ┐ 恢复中断屏蔽寄存器(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 al, 20h
        out     20h, al                 ; 发送 EOI

        mov ax, SelectorData
        mov ds, ax
        mov es, ax

        mov     ah, 0Fh                 ; 0000: 黑底    1111: 白字
        mov     al, '1'
    add eax, dword [ds:nowProcess]
        mov     [gs:((80 * 0 + 0) * 2)], ax     ; 屏幕第 0 行, 第 0 列。

        mov esi, dword [ds:nowProcess]
        shl esi, 4
        add     esi, 0Ch                ; 得到  word [ds:PCB+esi]中存放的当前进程的counter
        cmp word [ds:PCB+esi], 0
        jz findtorun
        dec word [ds:PCB+esi]
        iretd
findtorun:
        mov ecx, dword [ds: NumberOfProcess]
        xor esi, esi
        add     esi, 0Ch
```

```
CheckZeros:
        cmp word [ds:PCB+esi] , 0
        jne notAllZeros
        add esi, 010h
        dec ecx
        cmp ecx, 0
        jnz CheckZeros

        jmp allZeros
notAllZeros:
        xor bx, bx ; bx存放当前counter最大的进程号
        xor dx, dx ; dx存放当前遍历的进程号
        xor esi, esi
        add esi, 0ch
        xor edi, edi
        add edi, 0ch
        mov ecx, dword [ds: NumberOfProcess]
findMaxcountPro:
        mov ax, word [ds:PCB+esi]
        cmp word [ds:PCB+edi], ax
        ja getnewmax
        jmp nextmaxloop
getnewmax:
        mov bx, dx
        mov esi, edi
nextmaxloop:
        inc dx
        add edi, 10h
        dec ecx
        cmp ecx, 0
        jnz findMaxcountPro
        xor eax, eax
        mov ax, bx
        mov dword [ds:nowProcess], eax
        jmp continueThisProcess
allZeros:
        xor esi, esi
        add esi, 0ch
        mov ecx, dword [ds:NumberOfProcess]
copypri:
        mov bx, word [ds:PCB + esi + 2]
        mov word [ds:PCB + esi], bx
```

```
588         add esi, 10h
589         dec ecx
590         cmp ecx, 0
591         jnz copypri
592         jmp notAllZeros
593 
594 continueThisProcess:
595         mov esi, dword [ds:nowProcess]
596         shl esi, 4
597         add esi, 0Ch
598         dec word [ds:PCB+esi]
599 
600         mov ebx, dword [ds:nowProcess]
601         shl ebx, 3
602         jmp far [es:ebx + TSSArray]
603         iretd
604 
605 _UserIntHandler:
606 UserIntHandler  equ     _UserIntHandler - $$
607         mov     ah, 0Ch                         ; 0000: 黑底    1100: 红字
608         mov     al, 'I'
609         mov     [gs:((80 * 0 + 80) * 2)], ax    ; 屏幕第 0 行, 第 70 列。
610         iretd
611 
612 _SpuriousHandler:
613 SpuriousHandler equ     _SpuriousHandler - $$
614         mov     ah, 0Ch                         ; 0000: 黑底    1100: 红字
615         mov     al, '!'
616         mov     [gs:((80 * 0 + 70) * 2)], ax    ; 屏幕第 0 行, 第 75 列。
617         jmp     $
618         iretd
619 ; ------------------------------------------------------------------------
620 
621 ; 启动分页机制 ------------------------------------------------------------
622 SetupPaging:
623         ; 根据内存大小计算应初始化多少PDE以及多少页表
624         xor     edx, edx
625         mov     eax, [dwMemSize]
626         mov     ebx, 400000h    ; 400000h = 4M = 4096 * 1024, 一个页表对应的内存大小
627         div     ebx
628         mov     ecx, eax        ; 此时 ecx 为页表的个数，也即 PDE 应该的个数
629         test    edx, edx
```

```
        jz      .no_remainder
        inc     ecx             ; 如果余数不为 0 就需增加一个页表
.no_remainder:
        mov     [PageTableNumber], ecx  ; 暂存页表个数

        ; 初始化页表与页目录
        ; 为简化处理, 所有线性地址对应相等的物理地址. 并且不考虑内存空洞.
%assign                 num     1
%rep                    SetNumberOfProcess
        ; 首先初始化页目录
        mov     ax, SelectorFlatRW
        mov     es, ax
        mov     edi, PageDirBase%[num]  ; 此段首地址为 PageDirBase
        xor     eax, eax
        mov     eax, PageTblBase%[num] | PG_P  | PG_USU | PG_RWW
LocalTable1%[num]:
        stosd
        add     eax, 4096               ; 为了简化, 所有页表在内存中是连续的.
        loop    LocalTable1%[num]

        ; 再初始化所有页表
        mov     eax, [PageTableNumber]  ; 页表个数
        mov     ebx, 1024               ; 每个页表 1024 个 PTE
        mul     ebx
        mov     ecx, eax                ; PTE个数 = 页表个数 * 1024
        mov     edi, PageTblBase%[num]  ; 此段首地址为 PageTblBase
        xor     eax, eax
        mov     eax, PG_P  | PG_USU | PG_RWW
LocalTable2%[num]:
        stosd
        add     eax, 4096               ; 每一页指向 4K 的空间
        loop    LocalTable2%[num]

        ; 在此假设内存是大于 8M 的
        mov     eax, LinearAddrDemo
        shr     eax, 22
        mov     ebx, 4096
        mul     ebx
        mov     ecx, eax
        mov     eax, LinearAddrDemo
        shr     eax, 12
        and     eax, 03FFh      ; 1111111111b (10 bits)
```

```
672         mov     ebx, 4
673         mul     ebx
674         add     eax, ecx
675         add     eax, PageTblBase%[num]
676         mov     dword [es:eax], ProcTask%[num] | PG_P | PG_USU | PG_RWW
677 %assign                 num     num + 1
678 %endrep
679
680
681
682         mov     eax, PageDirBase1
683         mov     cr3, eax
684         mov     eax, cr0
685         or      eax, 80000000h
686         mov     cr0, eax
687         jmp     short .3
688 .3:
689         nop
690
691         ret
692 ; 分页机制启动完毕 ----------------------------------------------------------
693
694 ; 测试分页机制 --------------------------------------------------------------
695 StartPage:
696         mov     ax, cs
697         mov     ds, ax
698         mov     ax, SelectorFlatRW
699         mov     es, ax
700
701 %assign                 num     1
702 %rep                    SetNumberOfProcess
703         push    LenTask%[num]
704         push    OffsetTask%[num]
705         push    ProcTask%[num]
706         call    MemCpy
707         add     esp, 12
708 %assign                 num     num + 1
709 %endrep
710
711         push    LenPagingDemoAll
712         push    OffsetPagingDemoProc
713         push    ProcPagingDemo
```

```
714         call    MemCpy
715         add     esp, 12
716
717         mov     ax, SelectorData
718         mov     ds, ax                  ; 数据段选择子
719         mov     es, ax
720
721         call    SetupPaging             ; 启动分页
722
723         ret
724 ; ------------------------------------------------------------------------
725
726
727 ; PagingDemoProc --------------------------------------------------------
728 PagingDemoProc:
729 OffsetPagingDemoProc    equ     PagingDemoProc - $$
730         mov             eax, LinearAddrDemo
731         call    eax
732         retf
733 ; ------------------------------------------------------------------------
734 LenPagingDemoAll        equ     $ - PagingDemoProc
735 ; ------------------------------------------------------------------------
736
737
738 ; task1 ------------------------------------------------------------------------
739 task1:
740 OffsetTask1     equ     task1 - $$
741 looptask1:
742         mov     ah, 0Ch                 ; 0000: 黑底    1100: 红字
743         mov     al, 'V'
744         mov     [gs:((80 * 17 + 0) * 2)], ax    ; 屏幕第 17 行, 第 0 列。
745         mov     al, 'E'
746         mov     [gs:((80 * 17 + 1) * 2)], ax    ; 屏幕第 17 行, 第 1 列。
747         mov     al, 'R'
748         mov     [gs:((80 * 17 + 2) * 2)], ax    ; 屏幕第 17 行, 第 2 列。
749         mov     al, 'Y'
750         mov     [gs:((80 * 17 + 3) * 2)], ax    ; 屏幕第 17 行, 第 3 列。
751         jmp looptask1
752         ret
753 LenTask1        equ     $ - task1
754 ; ------------------------------------------------------------------------
755
```

```
756
757 ; task2 ------------------------------------------------------------------------
758 task2:
759 OffsetTask2     equ     task2 - $$
760 looptask2:
761         mov     ah, 0Fh                 ; 0000: 黑底    1111: 白字
762         mov     al, 'L'
763         mov     [gs:((80 * 17 + 0) * 2)], ax    ; 屏幕第 17 行, 第 0 列。
764         mov     al, 'O'
765         mov     [gs:((80 * 17 + 1) * 2)], ax    ; 屏幕第 17 行, 第 1 列。
766         mov     al, 'V'
767         mov     [gs:((80 * 17 + 2) * 2)], ax    ; 屏幕第 17 行, 第 2 列。
768         mov     al, 'E'
769         mov     [gs:((80 * 17 + 3) * 2)], ax    ; 屏幕第 17 行, 第 3 列。
770         jmp looptask2
771         ret
772 LenTask2        equ     $ - task2
773 ; ------------------------------------------------------------------------
774
775
776 ; task3 ------------------------------------------------------------------------
777 task3:
778 OffsetTask3     equ     task3 - $$
779 looptask3:
780         mov     ah, 0Ch                 ; 0000: 黑底    1100: 红字
781         mov     al, 'H'
782         mov     [gs:((80 * 17 + 0) * 2)], ax    ; 屏幕第 17 行, 第 0 列。
783         mov     al, 'U'
784         mov     [gs:((80 * 17 + 1) * 2)], ax    ; 屏幕第 17 行, 第 1 列。
785         mov     al, 'S'
786         mov     [gs:((80 * 17 + 2) * 2)], ax    ; 屏幕第 17 行, 第 2 列。
787         mov     al, 'T'
788         mov     [gs:((80 * 17 + 3) * 2)], ax    ; 屏幕第 17 行, 第 3 列。
789         jmp looptask3
790         ret
791 LenTask3        equ     $ - task3
792 ; ------------------------------------------------------------------------
793
794
795 ; task4 ------------------------------------------------------------------------
796 task4:
797 OffsetTask4     equ     task4 - $$
```

```
798 looptask4:
799         mov     ah, 0Fh                 ; 0000: 黑底    1111: 白字
800         mov     al, 'M'
801         mov     [gs:((80 * 17 + 0) * 2)], ax    ; 屏幕第 17 行, 第 0 列。
802         mov     al, 'R'
803         mov     [gs:((80 * 17 + 1) * 2)], ax    ; 屏幕第 17 行, 第 1 列。
804         mov     al, 'S'
805         mov     [gs:((80 * 17 + 2) * 2)], ax    ; 屏幕第 17 行, 第 2 列。
806         mov     al, 'U'
807         mov     [gs:((80 * 17 + 3) * 2)], ax    ; 屏幕第 17 行, 第 3 列。
808         jmp looptask4
809         ret
810 LenTask4        equ     $ - task4
811 ; ---------------------------------------------------------------------------
812
813 ; task5 ------------------------------------------------------------------------
814 task5:
815 OffsetTask5     equ     task5 - $$
816 looptask5:
817         mov     ah, 0Ch                 ; 0000: 黑底    1100: 红字
818         mov     al, '-'
819         mov     [gs:((80 * 17 + 0) * 2)], ax    ; 屏幕第 17 行, 第 0 列。
820         mov     al, 'H'
821         mov     [gs:((80 * 17 + 1) * 2)], ax    ; 屏幕第 17 行, 第 1 列。
822         mov     al, 'H'
823         mov     [gs:((80 * 17 + 2) * 2)], ax    ; 屏幕第 17 行, 第 2 列。
824         mov     al, 'F'
825         mov     [gs:((80 * 17 + 3) * 2)], ax    ; 屏幕第 17 行, 第 3 列。
826         jmp looptask5
827         ret
828 LenTask5        equ     $ - task5
829 ; ---------------------------------------------------------------------------
830
831 ; 显示内存信息 -------------------------------------------------------------
832 DispMemSize:
833         push    esi
834         push    edi
835         push    ecx
836
837         mov     esi, MemChkBuf
838         mov     ecx, [dwMCRNumber]      ;for(int i=0;i<[MCRNumber];i++) // 每次得到一个ARDS(Address Range Descriptor Structure)结构
839 .loop:                                  ;{
```

```
840         mov     edx, 5                          ;       for(int j=0;j<5;j++)    // 每次得到一个ARDS中的成员，共5个成员
841         mov     edi, ARDStruct          ;       {                       // 依次显示：BaseAddrLow，BaseAddrHigh，LengthLow，LengthHigh，Type
842 .1:                                                    ;
843         push    dword [esi]             ;
844         call    DispInt                 ;               DispInt(MemChkBuf[j*4]); // 显示一个成员
845         pop     eax                                     ;
846         stosd                                   ;               ARDStruct[j*4] = MemChkBuf[j*4];
847         add     esi, 4                          ;
848         dec     edx                                     ;
849         cmp     edx, 0                          ;
850         jnz     .1                                      ;       }
851         call    DispReturn              ;       printf("\n");
852         cmp     dword [dwType], 1       ;       if(Type == AddressRangeMemory) // AddressRangeMemory : 1, AddressRangeReserved : 2
853         jne     .2                                      ;       {
854         mov     eax, [dwBaseAddrLow];
855         add     eax, [dwLengthLow]      ;
856         cmp     eax, [dwMemSize]        ;               if(BaseAddrLow + LengthLow > MemSize)
857         jb      .2                                      ;
858         mov     [dwMemSize], eax        ;                       MemSize = BaseAddrLow + LengthLow;
859 .2:                                                     ;       }
860         loop    .loop                   ;}
861                                                         ;
862         call    DispReturn              ;printf("\n");
863         push    szRAMSize               ;
864         call    DispStr                 ;printf("RAM size:");
865         add     esp, 4                          ;
866                                                         ;
867         push    dword [dwMemSize];
868         call    DispInt                 ;DispInt(MemSize);
869         add     esp, 4                          ;
870
871         pop     ecx
872         pop     edi
873         pop     esi
874         ret
875 ; ------------------------------------------------------------------------
876
877 %include        "lib.inc"       ; 库函数
878
879 SegCode32Len    equ     $ - LABEL_SEG_CODE32
880 ; END of [SECTION .s32]
881
```

```
882
883 ; 16 位代码段. 由 32 位代码段跳入, 跳出后到实模式
884 [SECTION .s16code]
885 ALIGN   32
886 [BITS   16]
887 LABEL_SEG_CODE16:
888         ; 跳回实模式:
889         mov     ax, SelectorNormal
890         mov     ds, ax
891         mov     es, ax
892         mov     fs, ax
893         mov     gs, ax
894         mov     ss, ax
895
896         mov     eax, cr0
897         and     al, 11111110b
898         mov     cr0, eax
899
900 LABEL_GO_BACK_TO_REAL:
901         jmp     0:LABEL_REAL_ENTRY      ; 段地址会在程序开始处被设置成正确的值
902
903 Code16Len       equ     $ - LABEL_SEG_CODE16
904 ; END of [SECTION .s16code]
905
906
907 %assign                 num                                     1
908 %rep                    SetNumberOfProcess
909 ; TSS ----------------------------------------------------------------------------------------
910 ;初始化任务状态堆栈段(TSS)
911 [SECTION .tss]          ;求得各段的大小
912 ALIGN   32              ;align是一个让数据对齐的宏。通常align的对象是1、4、8等。这里的align 32是没有意义的，因为本来就是只有32b的地址总线宽
    度。
913 [BITS   32]             ;32位模式的机器运行
914 LABEL_TSS%[num]:              ;定义LABEL_TSS
915                 DD      0                       ; Back
916                 DD      TopOfStack%[num]        ; 0 级堆栈    //内层ring0级堆栈放入TSS中
917                 DD      SelectorStack%[num];
918                 DD      0                       ; 1 级堆栈
919                 DD      0                       ;
920                 DD      0                       ; 2 级堆栈
921                 DD      0                       ;               //TSS中最高只能放入Ring2级堆栈，ring3级堆栈不需要放入
922                 DD      PageDirBase%[num]; CR3
```

```
                DD      0                       ; EIP
                DD      0                       ; EFLAGS
                DD      0                       ; EAX
                DD      0                       ; ECX
                DD      0                       ; EDX
                DD      0                       ; EBX
                DD      Stack%[num]Len  ; ESP
                DD      0                       ; EBP
                DD      0                       ; ESI
                DD      0                       ; EDI
                DD      0                       ; ES
                DD      SelectorLDT%[num]Task                   ; CS
                DD      SelectorLDT%[num]Stack                  ; SS
                DD      SelectorLDT%[num]Data                   ; DS
                DD      0                       ; FS
                DD      SelectorVideo                   ; GS
                DD      SelectorLDT%[num]; LDT
                DW      0                       ; 调试陷阱标志
                DW      $ - LABEL_TSS%[num] + 2 ; I/O位图基址
                DB      0ffh                    ; I/O位图结束标志
TSSLen%[num]            equ     $ - LABEL_TSS%[num]   ;求得段的大小
; TSS ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


; LDT
[SECTION .ldt]
ALIGN   32
LABEL_LDT%[num]:
;                                         段基址       段界限     ,   属性
LABEL_LDT%[num]_DESC_TASK:      Descriptor             0,      Task%[num]Len - 1,   DA_C + DA_32 ; Code, 32 位
LABEL_LDT%[num]_DESC_DATA:      Descriptor             0,      Data%[num]Len - 1,   DA_DRW               ; Data, 32 位
LABEL_LDT%[num]_DESC_STACK:     Descriptor             0,                Stack%[num]Len,   DA_DRW + DA_32; Stack, 32 位
LDT%[num]Len            equ     $ - LABEL_LDT%[num]

; LDT 选择子
SelectorLDT%[num]Task   equ     LABEL_LDT%[num]_DESC_TASK - LABEL_LDT%[num] + SA_TIL
SelectorLDT%[num]Data   equ     LABEL_LDT%[num]_DESC_DATA - LABEL_LDT%[num] + SA_TIL
SelectorLDT%[num]Stack  equ     LABEL_LDT%[num]_DESC_STACK - LABEL_LDT%[num] + SA_TIL
; END of [SECTION .ldt]

; Task (LDT, 32 位代码段)
[SECTION .la]
```

```
ALIGN   32
[BITS   32]
LABEL_Task%[num]:
        sti
        call SelectorFlatC:ProcPagingDemo
        jmp     LABEL_Task%[num]
Task%[num]Len   equ     $ - LABEL_Task%[num]
; END of [SECTION .la]

; Data
[SECTION .da]
ALIGN   32
[BITS   32]
LABEL_Data%[num]:
Data%[num]Len   equ $ - LABEL_Data%[num]
; END of [SECTION .da]

[SECTION .sa]
ALIGN   32
[BITS   32]
Local_LABEL_STACK%[num]:
        times 512 db 0
Stack%[num]Len  equ     $ - Local_LABEL_STACK%[num] - 1
; END of [SECTION .sa]
%assign                 num                                     num + 1
%endrep


```