

```
1 ;代码清单5-1
2 ;文件名: c05_mbr.asm
3 ;文件说明: 硬盘主引导扇区代码
4 ;创建日期: 2011-3-31 21:15
5
6 mov ax,0xb800 ;指向文本模式的显示缓冲区
7 mov es,ax
8
9 ;以下显示字符串"Label offset:"
10 mov byte [es:0x00], 'L'
11 mov byte [es:0x01], 0x07
12 mov byte [es:0x02], 'a'
13 mov byte [es:0x03], 0x07
14 mov byte [es:0x04], 'b'
15 mov byte [es:0x05], 0x07
16 mov byte [es:0x06], 'e'
17 mov byte [es:0x07], 0x07
18 mov byte [es:0x08], 'l'
19 mov byte [es:0x09], 0x07
20 mov byte [es:0x0a], ' '
21 mov byte [es:0x0b], 0x07
22 mov byte [es:0x0c], "o"
23 mov byte [es:0x0d], 0x07
24 mov byte [es:0x0e], 'f'
25 mov byte [es:0x0f], 0x07
26 mov byte [es:0x10], 'f'
27 mov byte [es:0x11], 0x07
28 mov byte [es:0x12], 's'
29 mov byte [es:0x13], 0x07
30 mov byte [es:0x14], 'e'
31 mov byte [es:0x15], 0x07
32 mov byte [es:0x16], 't'
33 mov byte [es:0x17], 0x07
34 mov byte [es:0x18], ':'
35 mov byte [es:0x19], 0x07
36
37 mov ax,number ;取得标号number的偏移地址
38 mov bx,10
39
40 ;设置数据段的基地址
41 mov cx,cs
42 mov ds,cx
43
44 ;求个位上的数字
45 mov dx,0
46 div bx
47 mov [0x7c00+number+0x00],dl ;保存个位上的数字
48
49 ;求十位上的数字
50 xor dx,dx
51 div bx
52 mov [0x7c00+number+0x01],dl ;保存十位上的数字
53
54 ;求百位上的数字
55 xor dx,dx
56 div bx
57 mov [0x7c00+number+0x02],dl ;保存百位上的数字
58
59 ;求千位上的数字
60 xor dx,dx
61 div bx
62 mov [0x7c00+number+0x03],dl ;保存千位上的数字
63
64 ;求万位上的数字
65 xor dx,dx
66 div bx
67 mov [0x7c00+number+0x04],dl ;保存万位上的数字
68
69 ;以下用十进制显示标号的偏移地址
70 mov al,[0x7c00+number+0x04]
71 add al,0x30
72 mov [es:0x1a],al
73 mov byte [es:0x1b],0x04
74
75 mov al,[0x7c00+number+0x03]
76 add al,0x30
77 mov [es:0x1c],al
78 mov byte [es:0x1d],0x04
79
80 mov al,[0x7c00+number+0x02]
81 add al,0x30
82 mov [es:0x1e],al
83 mov byte [es:0x1f],0x04
84
85 mov al,[0x7c00+number+0x01]
86 add al,0x30
87 mov [es:0x20],al
88 mov byte [es:0x21],0x04
89
90 mov al,[0x7c00+number+0x00]
91 add al,0x30
92 mov [es:0x22],al
```

```
93         mov byte [es:0x23],0x04
94
95         mov byte [es:0x24],'D'
96         mov byte [es:0x25],0x07
97
98     infi: jmp near infi           ;无限循环
99
100 number db 0,0,0,0,0
101
102 times 203 db 0
103         db 0x55,0xaa
```

```
1      ;代码清单6-1
2      ;文件名: c06_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码
4      ;创建日期: 2011-4-12 22:12
5
6      jmp near start
7
8      mytext db 'L',0x07,'a',0x07,'b',0x07,'e',0x07,'l',0x07,' ',0x07,'o',0x07,\
9              'f',0x07,'f',0x07,'s',0x07,'e',0x07,'t',0x07,':',0x07
10     number db 0,0,0,0
11
12     start:
13         mov ax,0x7c0             ;设置数据段基地址
14         mov ds,ax
15
16         mov ax,0xb800           ;设置附加段基地址
17         mov es,ax
18
19         cld
20         mov si,mytext
21         mov di,0
22         mov cx,(number-mytext)/2 ;实际上等于 13
23         rep movsw
24
25         ;得到标号所代表的偏移地址
26         mov ax,number
27
28         ;计算各个数位
29         mov bx,ax
30         mov cx,5                 ;循环次数
31         mov si,10                ;除数
32     digit:
33         xor dx,dx
34         div si
35         mov [bx],dl              ;保存数位
36         inc bx
37         loop digit
38
39         ;显示各个数位
40         mov bx,number
41         mov si,4
42     show:
43         mov al,[bx+si]
44         add al,0x30
45         mov ah,0x04
46         mov [es:di],ax
47         add di,2
48         dec si
49         jns show
50
51         mov word [es:di],0x0744
52
53         jmp near $
54
55     times 510-($-$$) db 0
56                     db 0x55,0xaa
```



```
1      ;代码清单7-1
2      ;文件名: c07_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码
4      ;创建日期: 2011-4-13 18:02
5
6      jmp near start
7
8      message db '1+2+3+...+100='
9
10     start:
11         mov ax,0x7c0      ;设置数据段的段基地址
12         mov ds,ax
13
14         mov ax,0xb800     ;设置附加段基址到显示缓冲区
15         mov es,ax
16
17         ;以下显示字符串
18         mov si,message
19         mov di,0
20         mov cx,start-message
21     @g:
22         mov al,[si]
23         mov [es:di],al
24         inc di
25         mov byte [es:di],0x07
26         inc di
27         inc si
28         loop @g
29
30         ;以下计算1到100的和
31         xor ax,ax
32         mov cx,1
33     @f:
34         add ax,cx
35         inc cx
36         cmp cx,100
37         jle @f
38
39         ;以下计算累加和的每个数位
40         xor cx,cx      ;设置堆栈段的段基地址
41         mov ss,cx
42         mov sp,cx
43
44         mov bx,10
45         xor cx,cx
46     @d:
47         inc cx
48         xor dx,dx
49         div bx
50         or dl,0x30
51         push dx
52         cmp ax,0
53         jne @d
54
55         ;以下显示各个数位
56     @a:
57         pop dx
58         mov [es:di],dl
59         inc di
60         mov byte [es:di],0x07
61         inc di
62         loop @a
63
64         jmp near $
65
66     times 510-($-$$) db 0
67                     db 0x55,0xaa
68
```



```

1      ;代码清单8-2
2      ;文件名: c08.asm
3      ;文件说明: 用户程序
4      ;创建日期: 2011-5-5 18:17
5
6      ;=====
7      SECTION header vstart=0          ;定义用户程序头部段
8      program_length dd program_end    ;程序总长度[0x00]
9
10     ;用户程序入口点
11     code_entry      dw start          ;偏移地址[0x04]
12                     dd section.code_1.start ;段地址[0x06]
13
14     realloc_tbl_len dw (header_end-code_1_segment)/4
15                     ;段重定位表项个数[0x0a]
16
17     ;段重定位表
18     code_1_segment  dd section.code_1.start ;[0x0c]
19     code_2_segment  dd section.code_2.start ;[0x10]
20     data_1_segment  dd section.data_1.start ;[0x14]
21     data_2_segment  dd section.data_2.start ;[0x18]
22     stack_segment   dd section.stack.start  ;[0x1c]
23
24     header_end:
25
26     ;=====
27     SECTION code_1 align=16 vstart=0    ;定义代码段1 (16字节对齐)
28     put_string:                          ;显示串(0结尾)。
29                                         ;输入: DS:BX=串地址
30
31     mov cl,[bx]
32     or cl,cl                             ;cl=0 ?
33     jz .exit                             ;是的, 返回主程序
34     call put_char
35     inc bx                               ;下一个字符
36     jmp put_string
37
38     .exit:
39     ret
40
41     ;-----
42     put_char:                            ;显示一个字符
43                                         ;输入: cl=字符ascii
44
45     push ax
46     push bx
47     push cx
48     push dx
49     push ds
50     push es
51
52     ;以下取当前光标位置
53     mov dx,0x3d4
54     mov al,0x0e
55     out dx,al
56     mov dx,0x3d5
57     in al,dx                             ;高8位
58     mov ah,al
59
60     mov dx,0x3d4
61     mov al,0x0f
62     out dx,al
63     mov dx,0x3d5
64     in al,dx                             ;低8位
65     mov bx,ax                            ;BX=代表光标位置的16位数
66
67     cmp cl,0x0d                          ;回车符?
68     jnz .put_0a                          ;不是。看看是不是换行等字符
69     mov ax,bx                            ;此句略显多余, 但去掉后还得改书, 麻烦
70     mov bl,80
71     div bl
72     mul bl
73     mov bx,ax
74     jmp .set_cursor
75
76     .put_0a:
77     cmp cl,0x0a                          ;换行符?
78     jnz .put_other                       ;不是, 那就正常显示字符
79     add bx,80
80     jmp .roll_screen
81
82     .put_other:                          ;正常显示字符
83     mov ax,0xb800
84     mov es,ax
85     shl bx,1
86     mov [es:bx],cl
87
88     ;以下将光标位置推进一个字符
89     shr bx,1
90     add bx,1
91
92     .roll_screen:
93     cmp bx,2000                          ;光标超出屏幕? 滚屏
94     jl .set_cursor

```

```

93
94     mov ax,0xb800
95     mov ds,ax
96     mov es,ax
97     cld
98     mov si,0xa0
99     mov di,0x00
100    mov cx,1920
101    rep movsw
102    mov bx,3840                ;清除屏幕最底一行
103    mov cx,80
104 .cls:
105     mov word[es:bx],0x0720
106     add bx,2
107     loop .cls
108
109     mov bx,1920
110
111 .set_cursor:
112     mov dx,0x3d4
113     mov al,0x0e
114     out dx,al
115     mov dx,0x3d5
116     mov al,bh
117     out dx,al
118     mov dx,0x3d4
119     mov al,0x0f
120     out dx,al
121     mov dx,0x3d5
122     mov al,bl
123     out dx,al
124
125     pop es
126     pop ds
127     pop dx
128     pop cx
129     pop bx
130     pop ax
131
132     ret
133
134 ;-----
135 start:
136     ;初始执行时, DS和ES指向用户程序头部段
137     mov ax,[stack_segment]    ;设置到用户程序自己的堆栈
138     mov ss,ax
139     mov sp,stack_end
140
141     mov ax,[data_1_segment]    ;设置到用户程序自己的数据段
142     mov ds,ax
143
144     mov bx,msg0
145     call put_string           ;显示第一段信息
146
147     push word [es:code_2_segment]
148     mov ax,begin
149     push ax                   ;可以直接push begin,80386+
150
151     retf                      ;转移到代码段2执行
152
153 continue:
154     mov ax,[es:data_2_segment] ;段寄存器DS切换到数据段2
155     mov ds,ax
156
157     mov bx,msg1
158     call put_string           ;显示第二段信息
159
160     jmp $
161
162 ;=====
163 SECTION code_2 align=16 vstart=0 ;定义代码段2 (16字节对齐)
164
165 begin:
166     push word [es:code_1_segment]
167     mov ax,continue
168     push ax                   ;可以直接push continue,80386+
169
170     retf                      ;转移到代码段1接着执行
171
172 ;=====
173 SECTION data_1 align=16 vstart=0
174
175 msg0 db ' This is NASM - the famous Netwide Assembler. '
176     db 'Back at SourceForge and in intensive development! '
177     db 'Get the current versions from http://www.nasm.us/. '
178     db 0x0d,0x0a,0x0d,0x0a
179     db ' Example code for calculate 1+2+...+1000:',0x0d,0x0a,0x0d,0x0a
180     db '     xor dx,dx',0x0d,0x0a
181     db '     xor ax,ax',0x0d,0x0a
182     db '     xor cx,cx',0x0d,0x0a
183     db ' @@@:',0x0d,0x0a
184     db '     inc cx',0x0d,0x0a

```



```
185         db '      add ax,cx',0x0d,0x0a
186         db '      adc dx,0',0x0d,0x0a
187         db '      inc cx',0x0d,0x0a
188         db '      cmp cx,1000',0x0d,0x0a
189         db '      jle @@',0x0d,0x0a
190         db '      ... ...(Some other codes)',0x0d,0x0a,0x0d,0x0a
191         db 0
192
193         ;=====
194         SECTION data_2 align=16 vstart=0
195
196         msg1 db ' The above contents is written by LeeChung. '
197             db '2011-05-06'
198             db 0
199
200         ;=====
201         SECTION stack align=16 vstart=0
202
203             resb 256
204
205         stack_end:
206
207         ;=====
208         SECTION trail align=16
209         program_end:
```



```
1      ;代码清单8-1
2      ;文件名: c08_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码(加载程序)
4      ;创建日期: 2011-5-5 18:17
5
6      app_lba_start equ 100          ;声明常数(用户程序起始逻辑扇区号)
7                                      ;常数的声明不会占用汇编地址
8
9      SECTION mbr align=16 vstart=0x7c00
10
11      ;设置堆栈段和栈指针
12      mov ax,0
13      mov ss,ax
14      mov sp,ax
15
16      mov ax,[cs:phy_base]          ;计算用于加载用户程序的逻辑段地址
17      mov dx,[cs:phy_base+0x02]
18      mov bx,16
19      div bx
20      mov ds,ax                    ;令DS和ES指向该段以进行操作
21      mov es,ax
22
23      ;以下读取程序的起始部分
24      xor di,di
25      mov si,app_lba_start          ;程序在硬盘上的起始逻辑扇区号
26      xor bx,bx                    ;加载到DS:0x0000处
27      call read_hard_disk_0
28
29      ;以下判断整个程序有多大
30      mov dx,[2]                    ;曾经把dx写成了ds, 花了二十分钟排错
31      mov ax,[0]
32      mov bx,512                    ;512字节每扇区
33      div bx
34      cmp dx,0
35      jnz @1                        ;未除尽, 因此结果比实际扇区数少1
36      dec ax                        ;已经读了一个扇区, 扇区总数减1
37      @1:
38      cmp ax,0                      ;考虑实际长度小于等于512个字节的情况
39      jz direct
40
41      ;读取剩余的扇区
42      push ds                        ;以下要用到并改变DS寄存器
43
44      mov cx,ax                      ;循环次数(剩余扇区数)
45      @2:
46      mov ax,ds
47      add ax,0x20                    ;得到下一个以512字节为边界的段地址
48      mov ds,ax
49
50      xor bx,bx                      ;每次读时, 偏移地址始终为0x0000
51      inc si                          ;下一个逻辑扇区
52      call read_hard_disk_0
53      loop @2                        ;循环读, 直到读完整个功能程序
54
55      pop ds                          ;恢复数据段基址到用户程序头部段
56
57      ;计算入口点代码段基址
58      direct:
59      mov dx,[0x08]
60      mov ax,[0x06]
61      call calc_segment_base
62      mov [0x06],ax                ;回填修正后的入口点代码段基址
63
64      ;开始处理段重定位表
65      mov cx,[0x0a]                ;需要重定位的项目数量
66      mov bx,0x0c                  ;重定位表首地址
67
68      realloc:
69      mov dx,[bx+0x02]              ;32位地址的高16位
70      mov ax,[bx]
71      call calc_segment_base
72      mov [bx],ax                  ;回填段的基址
73      add bx,4                      ;下一个重定位项(每项占4个字节)
74      loop realloc
75
76      jmp far [0x04]                ;转移到用户程序
77
78      ;-----
79      read_hard_disk_0:              ;从硬盘读取一个逻辑扇区
80                                      ;输入: DI:SI=起始逻辑扇区号
81                                      ;      DS:BX=目标缓冲区地址
82
83      push ax
84      push bx
85      push cx
86      push dx
87
88      mov dx,0x1f2
89      mov al,1
90      out dx,al                    ;读取的扇区数
91
92      inc dx                        ;0x1f3
93      mov ax,si
```

```
93         out dx,al                ;LBA地址7~0
94
95         inc dx                    ;0x1f4
96         mov al,ah
97         out dx,al                ;LBA地址15~8
98
99         inc dx                    ;0x1f5
100        mov ax,di
101        out dx,al                ;LBA地址23~16
102
103        inc dx                    ;0x1f6
104        mov al,0xe0                ;LBA28模式, 主盘
105        or al,ah                  ;LBA地址27~24
106        out dx,al
107
108        inc dx                    ;0x1f7
109        mov al,0x20                ;读命令
110        out dx,al
111
112        .waits:
113        in al,dx
114        and al,0x88
115        cmp al,0x08
116        jnz .waits                ;不忙, 且硬盘已准备好数据传输
117
118        mov cx,256                ;总共要读取的字数
119        mov dx,0x1f0
120        .readw:
121        in ax,dx
122        mov [bx],ax
123        add bx,2
124        loop .readw
125
126        pop dx
127        pop cx
128        pop bx
129        pop ax
130
131        ret
132
133        ;-----
134        calc_segment_base:        ;计算16位段地址
135                                   ;输入: DX:AX=32位物理地址
136                                   ;返回: AX=16位段基地址
137
138        push dx
139
140        add ax,[cs:phy_base]
141        adc dx,[cs:phy_base+0x02]
142        shr ax,4
143        ror dx,4
144        and dx,0xf000
145        or ax,dx
146
147        pop dx
148
149        ret
150
151        ;-----
152        phy_base dd 0x10000        ;用户程序被加载的物理起始地址
153
154        times 510-($-$$) db 0
155                                   db 0x55,0xaa
```

```
1      ;代码清单9-1
2      ;文件名: c09_1.asm
3      ;文件说明: 用户程序
4      ;创建日期: 2011-4-16 22:03
5
6      ;=====
7      SECTION header vstart=0      ;定义用户程序头部段
8      program_length dd program_end ;程序总长度[0x00]
9
10     ;用户程序入口点
11     code_entry      dw start      ;偏移地址[0x04]
12                   dd section.code.start ;段地址[0x06]
13
14     realloc_tbl_len dw (header_end-realloc_begin)/4
15                   ;段重定位表项个数[0x0a]
16
17     realloc_begin:
18     ;段重定位表
19     code_segment   dd section.code.start ;[0x0c]
20     data_segment   dd section.data.start ;[0x14]
21     stack_segment  dd section.stack.start ;[0x1c]
22
23     header_end:
24
25     ;=====
26     SECTION code align=16 vstart=0 ;定义代码段(16字节对齐)
27     new_int_0x70:
28         push ax
29         push bx
30         push cx
31         push dx
32         push es
33
34     .w0:
35         mov al,0x0a      ;阻断NMI。当然, 通常是不必要的
36         or al,0x80
37         out 0x70,al
38         in al,0x71      ;读寄存器A
39         test al,0x80    ;测试第7位UIP
40         jnz .w0         ;以上代码对于更新周期结束中断来说
41                   ;是不必要的
42
43         xor al,al
44         or al,0x80
45         out 0x70,al
46         in al,0x71      ;读RTC当前时间(秒)
47         push ax
48
49         mov al,2
50         or al,0x80
51         out 0x70,al
52         in al,0x71      ;读RTC当前时间(分)
53         push ax
54
55         mov al,4
56         or al,0x80
57         out 0x70,al
58         in al,0x71      ;读RTC当前时间(时)
59         push ax
60
61         mov al,0x0c     ;寄存器C的索引。且开放NMI
62         out 0x70,al
63         in al,0x71      ;读一下RTC的寄存器C, 否则只发生一次中断
64                   ;此处不考虑闹钟和周期性中断的情况
65         mov ax,0xb800
66         mov es,ax
67
68         pop ax
69         call bcd_to_ascii
70         mov bx,12*160 + 36*2 ;从屏幕上的12行36列开始显示
71
72         mov [es:bx],ah
73         mov [es:bx+2],al ;显示两位小时数字
74
75         mov al,':'
76         mov [es:bx+4],al ;显示分隔符':'
77         not byte [es:bx+5] ;反转显示属性
78
79         pop ax
80         call bcd_to_ascii
81         mov [es:bx+6],ah
82         mov [es:bx+8],al ;显示两位分钟数字
83
84         mov al,':'
85         mov [es:bx+10],al ;显示分隔符':'
86         not byte [es:bx+11] ;反转显示属性
87
88         pop ax
89         call bcd_to_ascii
90         mov [es:bx+12],ah
91         mov [es:bx+14],al ;显示两位小时数字
92
93         mov al,0x20      ;中断结束命令EOI
```

```

93      out 0xa0,a1          ;向从片发送
94      out 0x20,a1          ;向主片发送
95
96      pop es
97      pop dx
98      pop cx
99      pop bx
100     pop ax
101
102     iret
103
104     ;-----
105     bcd_to_ascii:          ;BCD码转ASCII
106                             ;输入: AL=bcd码
107                             ;输出: AX=ascii
108     mov ah,al              ;分拆成两个数字
109     and al,0x0f            ;仅保留低4位
110     add al,0x30            ;转换成ASCII
111
112     shr ah,4               ;逻辑右移4位
113     and ah,0x0f
114     add ah,0x30
115
116     ret
117
118     ;-----
119     start:
120     mov ax,[stack_segment]
121     mov ss,ax
122     mov sp,ss_pointer
123     mov ax,[data_segment]
124     mov ds,ax
125
126     mov bx,init_msg        ;显示初始信息
127     call put_string
128
129     mov bx,inst_msg        ;显示安装信息
130     call put_string
131
132     mov al,0x70
133     mov bl,4
134     mul bl                  ;计算0x70号中断在IVT中的偏移
135     mov bx,ax
136
137     cli                    ;防止改动期间发生新的0x70号中断
138
139     push es
140     mov ax,0x0000
141     mov es,ax
142     mov word [es:bx],new_int_0x70 ;偏移地址。
143
144     mov word [es:bx+2],cs   ;段地址
145     pop es
146
147     mov al,0x0b            ;RTC寄存器B
148     or al,0x80             ;阻断NMI
149     out 0x70,al
150     mov al,0x12            ;设置寄存器B, 禁止周期性中断, 开放更
151     out 0x71,al            ;新结束后中断, BCD码, 24小时制
152
153     mov al,0x0c
154     out 0x70,al
155     in al,0x71             ;读RTC寄存器C, 复位未决的中断状态
156
157     in al,0xa1             ;读8259从片的IMR寄存器
158     and al,0xfe            ;清除bit 0(此位连接RTC)
159     out 0xa1,al            ;写回此寄存器
160
161     sti                    ;重新开放中断
162
163     mov bx,done_msg        ;显示安装完成信息
164     call put_string
165
166     mov bx,tips_msg        ;显示提示信息
167     call put_string
168
169     mov cx,0xb800
170     mov ds,cx
171     mov byte [12*160 + 33*2], '@' ;屏幕第12行, 35列
172
173     .idle:
174     hlt                    ;使CPU进入低功耗状态, 直到用中断唤醒
175     not byte [12*160 + 33*2+1] ;反转显示属性
176     jmp .idle
177
178     ;-----
179     put_string:            ;显示串(0结尾)。
180                             ;输入: DS:BX=串地址
181     mov cl,[bx]
182     or cl,cl                ;cl=0 ?
183     jz .exit                ;是的, 返回主程序
184     call put_char

```

```

185         inc bx                ;下一个字符
186         jmp put_string
187
188     .exit:
189         ret
190
191 ;-----
192 put_char:                ;显示一个字符
193                         ;输入: cl=字符ascii
194         push ax
195         push bx
196         push cx
197         push dx
198         push ds
199         push es
200
201         ;以下取当前光标位置
202         mov dx,0x3d4
203         mov al,0x0e
204         out dx,al
205         mov dx,0x3d5
206         in al,dx            ;高8位
207         mov ah,al
208
209         mov dx,0x3d4
210         mov al,0x0f
211         out dx,al
212         mov dx,0x3d5
213         in al,dx            ;低8位
214         mov bx,ax           ;BX=代表光标位置的16位数
215
216         cmp cl,0x0d         ;回车符?
217         jnz .put_0a         ;不是。看看是不是换行等字符
218         mov ax,bx           ;
219         mov bl,80
220         div bl
221         mul bl
222         mov bx,ax
223         jmp .set_cursor
224
225 .put_0a:
226         cmp cl,0x0a         ;换行符?
227         jnz .put_other      ;不是, 那就正常显示字符
228         add bx,80
229         jmp .roll_screen
230
231 .put_other:                ;正常显示字符
232         mov ax,0xb800
233         mov es,ax
234         shl bx,1
235         mov [es:bx],cl
236
237         ;以下将光标位置推进一个字符
238         shr bx,1
239         add bx,1
240
241 .roll_screen:
242         cmp bx,2000         ;光标超出屏幕? 滚屏
243         jl .set_cursor
244
245         mov ax,0xb800
246         mov ds,ax
247         mov es,ax
248         cld
249         mov si,0xa0
250         mov di,0x00
251         mov cx,1920
252         rep movsw
253         mov bx,3840         ;清除屏幕最底一行
254         mov cx,80
255 .cls:
256         mov word[es:bx],0x0720
257         add bx,2
258         loop .cls
259
260         mov bx,1920
261
262 .set_cursor:
263         mov dx,0x3d4
264         mov al,0x0e
265         out dx,al
266         mov dx,0x3d5
267         mov al,bh
268         out dx,al
269         mov dx,0x3d4
270         mov al,0x0f
271         out dx,al
272         mov dx,0x3d5
273         mov al,bl
274         out dx,al
275
276         pop es

```

```
277         pop ds
278         pop dx
279         pop cx
280         pop bx
281         pop ax
282
283         ret
284
285 ;=====
286 SECTION data align=16 vstart=0
287
288         init_msg      db 'Starting...',0x0d,0x0a,0
289
290         inst_msg       db 'Installing a new interrupt 70H...',0
291
292         done_msg       db 'Done.',0x0d,0x0a,0
293
294         tips_msg       db 'Clock is now working.',0
295
296 ;=====
297 SECTION stack align=16 vstart=0
298
299         resb 256
300 ss_pointer:
301
302 ;=====
303 SECTION program_trail
304 program_end:
```



```
1      ;代码清单9-2
2      ;文件名: c09_2.asm
3      ;文件说明: 用于演示BIOS中断的用户程序
4      ;创建日期: 2012-3-28 20:35
5
6      ;=====
7      SECTION header vstart=0                ;定义用户程序头部段
8          program_length dd program_end      ;程序总长度[0x00]
9
10     ;用户程序入口点
11     code_entry dw start                    ;偏移地址[0x04]
12                dd section.code.start      ;段地址[0x06]
13
14     realloc_tbl_len dw (header_end-realloc_begin)/4
15                                ;段重定位表项个数[0x0a]
16
17     realloc_begin:
18     ;段重定位表
19     code_segment dd section.code.start    ;[0x0c]
20     data_segment dd section.data.start    ;[0x14]
21     stack_segment dd section.stack.start  ;[0x1c]
22
23     header_end:
24
25     ;=====
26     SECTION code align=16 vstart=0          ;定义代码段(16字节对齐)
27     start:
28         mov ax,[stack_segment]
29         mov ss,ax
30         mov sp,ss_pointer
31         mov ax,[data_segment]
32         mov ds,ax
33
34         mov cx,msg_end-message
35         mov bx,message
36
37     .putc:
38         mov ah,0x0e
39         mov al,[bx]
40         int 0x10
41         inc bx
42         loop .putc
43
44     .reps:
45         mov ah,0x00
46         int 0x16
47
48         mov ah,0x0e
49         mov bl,0x07
50         int 0x10
51
52         jmp .reps
53
54     ;=====
55     SECTION data align=16 vstart=0
56
57     message db 'Hello, friend!',0x0d,0x0a
58             db 'This simple procedure used to demonstrate '
59             db 'the BIOS interrupt.',0x0d,0x0a
60             db 'Please press the keys on the keyboard ->'
61     msg_end:
62
63     ;=====
64     SECTION stack align=16 vstart=0
65
66         resb 256
67     ss_pointer:
68
69     ;=====
70     SECTION program_trail
71     program_end:
```



```
1      ;代码清单11-1
2      ;文件名: c11_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码
4      ;创建日期: 2011-5-16 19:54
5
6      ;设置堆栈段和栈指针
7      mov ax,cs
8      mov ss,ax
9      mov sp,0x7c00
10
11     ;计算GDT所在的逻辑段地址
12     mov ax,[cs:gdt_base+0x7c00]      ;低16位
13     mov dx,[cs:gdt_base+0x7c00+0x02] ;高16位
14     mov bx,16
15     div bx
16     mov ds,ax      ;令DS指向该段以进行操作
17     mov bx,dx      ;段内起始偏移地址
18
19     ;创建0#描述符,它是空描述符,这是处理器的要求
20     mov dword [bx+0x00],0x00
21     mov dword [bx+0x04],0x00
22
23     ;创建#1描述符,保护模式下的代码段描述符
24     mov dword [bx+0x08],0x7c0001ff
25     mov dword [bx+0x0c],0x00409800
26
27     ;创建#2描述符,保护模式下的数据段描述符(文本模式下的显示缓冲区)
28     mov dword [bx+0x10],0x8000ffff
29     mov dword [bx+0x14],0x0040920b
30
31     ;创建#3描述符,保护模式下的堆栈段描述符
32     mov dword [bx+0x18],0x00007a00
33     mov dword [bx+0x1c],0x00409600
34
35     ;初始化描述符表寄存器GDTR
36     mov word [cs: gdt_size+0x7c00],31 ;描述符表的界限(总字节数减一)
37
38     lgdt [cs: gdt_size+0x7c00]
39
40     in al,0x92      ;南桥芯片内的端口
41     or al,0000_0010B
42     out 0x92,al     ;打开A20
43
44     cli      ;保护模式下中断机制尚未建立,应
45             ;禁止中断
46
47     mov eax,cr0
48     or eax,1
49     mov cr0,eax    ;设置PE位
50
51     ;以下进入保护模式...
52     jmp dword 0x008:flush      ;16位的描述符选择子: 32位偏移
53                                 ;流水线并串行化处理
54     [bits 32]
55
56     flush:
57     mov cx,0000000000_10_000B ;加载数据段选择子(0x10)
58     mov ds,cx
59
60     ;以下在屏幕上显示"Protect mode OK."
61     mov byte [0x00],'P'
62     mov byte [0x02],'r'
63     mov byte [0x04],'o'
64     mov byte [0x06],'t'
65     mov byte [0x08],'e'
66     mov byte [0x0a],'c'
67     mov byte [0x0c],'t'
68     mov byte [0x0e],' '
69     mov byte [0x10],'m'
70     mov byte [0x12],'o'
71     mov byte [0x14],'d'
72     mov byte [0x16],'e'
73     mov byte [0x18],' '
74     mov byte [0x1a],'O'
75     mov byte [0x1c],'K'
76
77     ;以下用简单的示例来帮助阐述32位保护模式下的堆栈操作
78     mov cx,0000000000_11_000B ;加载堆栈段选择子
79     mov ss,cx
80     mov esp,0x7c00
81
82     mov ebp,esp ;保存堆栈指针
83     push byte '.' ;压入立即数(字节)
84
85     sub ebp,4
86     cmp ebp,esp ;判断压入立即数时,ESP是否减4
87     jnz ghalt
88     pop eax
89     mov [0x1e],al ;显示句点
90
91     ghalt:
92     hlt ;已经禁止中断,将不会被唤醒
```

```
93 ;-----  
94  
95     gdt_size      dw 0  
96     gdt_base      dd 0x00007e00    ;GDT的物理地址  
97  
98     times 510-($-$$) db 0  
99                     db 0x55,0xaa
```

```

1      ;代码清单12-1
2      ;文件名: c12_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码
4      ;创建日期: 2011-10-27 22:52
5
6      ;设置堆栈段和栈指针
7      mov eax,cs
8      mov ss,eax
9      mov sp,0x7c00
10
11     ;计算GDT所在的逻辑段地址
12     mov eax,[cs:pgdt+0x7c00+0x02]      ;GDT的32位线性基地址
13     xor edx,edx
14     mov ebx,16
15     div ebx                             ;分解成16位逻辑地址
16
17     mov ds,eax                          ;令DS指向该段以进行操作
18     mov ebx,edx                         ;段内起始偏移地址
19
20     ;创建0#描述符,它是空描述符,这是处理器的要求
21     mov dword [ebx+0x00],0x00000000
22     mov dword [ebx+0x04],0x00000000
23
24     ;创建1#描述符,这是一个数据段,对应0~4GB的线性地址空间
25     mov dword [ebx+0x08],0x0000ffff      ;基地址为0,段界限为0xfffff
26     mov dword [ebx+0x0c],0x00cf9200      ;粒度为4KB,存储器段描述符
27
28     ;创建保护模式下初始代码段描述符
29     mov dword [ebx+0x10],0x7c0001ff      ;基地址为0x00007c00,512字节
30     mov dword [ebx+0x14],0x00409800      ;粒度为1个字节,代码段描述符
31
32     ;创建以上代码段的别名描述符
33     mov dword [ebx+0x18],0x7c0001ff      ;基地址为0x00007c00,512字节
34     mov dword [ebx+0x1c],0x00409200      ;粒度为1个字节,数据段描述符
35
36     mov dword [ebx+0x20],0x7c00fffe
37     mov dword [ebx+0x24],0x00cf9600
38
39     ;初始化描述符表寄存器GDTR
40     mov word [cs:pgdt+0x7c00],39         ;描述符表的界限
41
42     lgdt [cs:pgdt+0x7c00]
43
44     in al,0x92                           ;南桥芯片内的端口
45     or al,0000_0010B
46     out 0x92,al                          ;打开A20
47
48     cli                                  ;中断机制尚未工作
49
50     mov eax,cr0
51     or eax,1
52     mov cr0,eax                          ;设置PE位
53
54     ;以下进入保护模式...
55     jmp dword 0x0010:flush               ;16位的描述符选择子:32位偏移
56
57     [bits 32]
58 flush:
59     mov eax,0x0018
60     mov ds,eax
61
62     mov eax,0x0008                       ;加载数据段(0..4GB)选择子
63     mov es,eax
64     mov fs,eax
65     mov gs,eax
66
67     mov eax,0x0020                       ;0000 0000 0010 0000
68     mov ss,eax
69     xor esp,esp                          ;ESP <- 0
70
71     mov dword [es:0x0b8000],0x072e0750 ;字符'P'、'.'及其显示属性
72     mov dword [es:0x0b8004],0x072e074d ;字符'M'、'.'及其显示属性
73     mov dword [es:0x0b8008],0x07200720 ;两个空白字符及其显示属性
74     mov dword [es:0x0b800c],0x076b076f ;字符'o'、'k'及其显示属性
75
76     ;开始冒泡排序
77     mov ecx,pgdt-string-1                ;遍历次数=串长度-1
78
79     @@1:
80     push ecx                             ;32位模式下的loop使用ecx
81     xor bx,bx                            ;32位模式下,偏移量可以是16位,也可以是后面的32位
82
83     @@2:
84     mov ax,[string+bx]
85     cmp ah,al
86     jge @@3
87     xchg al,ah
88     mov [string+bx],ax
89
90     @@3:
91     inc bx
92     loop @@2
93     pop ecx
94     loop @@1

```

```
93      mov ecx,pgdt-string
94      xor ebx,ebx                      ;偏移地址是32位的情况
95      @@4:                          ;32位的偏移具有更大的灵活性
96      mov ah,0x07
97      mov al,[string+ebx]
98      mov [es:0xb80a0+ebx*2],ax      ;演示0~4GB寻址。
99      inc ebx
100     loop @@4
101
102     hlt
103
104 ;-----
105     string      db 's0ke4or92xap3fv8giuzjcy5l1m7hd6bnqtw.'
106 ;-----
107     pgdt        dw 0
108                dd 0x00007e00      ;GDT的物理地址
109 ;-----
110     times 510-($-$$) db 0
111                db 0x55,0xaa
```

```
1 ;代码清单13-3
2 ;文件名: c13.asm
3 ;文件说明: 用户程序
4 ;创建日期: 2011-10-30 15:19
5
6 ;=====
7 SECTION header vstart=0
8
9     program_length dd program_end ;程序总长度#0x00
10
11     head_len dd header_end ;程序头部的长度#0x04
12
13     stack_seg dd 0 ;用于接收堆栈段选择子#0x08
14     stack_len dd 1 ;程序建议的堆栈大小#0x0c
15 ;以4KB为单位
16
17     prgentry dd start ;程序入口#0x10
18     code_seg dd section.code.start ;代码段位置#0x14
19     code_len dd code_end ;代码段长度#0x18
20
21     data_seg dd section.data.start ;数据段位置#0x1c
22     data_len dd data_end ;数据段长度#0x20
23
24 ;-----
25 ;符号地址检索表
26 salt_items dd (header_end-salt)/256 ;#0x24
27
28 salt: ;#0x28
29 PrintString db '@PrintString'
30             times 256-($-PrintString) db 0
31
32 TerminateProgram db '@TerminateProgram'
33                 times 256-($-TerminateProgram) db 0
34
35 ReadDiskData db '@ReadDiskData'
36              times 256-($-ReadDiskData) db 0
37
38 header_end:
39
40 ;=====
41 SECTION data vstart=0
42
43     buffer times 1024 db 0 ;缓冲区
44
45     message_1 db 0x0d,0x0a,0x0d,0x0a
46               db '*****User program is runing*****'
47               db 0x0d,0x0a,0
48     message_2 db ' Disk data:',0x0d,0x0a,0
49
50 data_end:
51
52 ;=====
53 [bits 32]
54 ;=====
55 SECTION code vstart=0
56 start:
57     mov eax,ds
58     mov fs,eax
59
60     mov eax,[stack_seg]
61     mov ss,eax
62     mov esp,0
63
64     mov eax,[data_seg]
65     mov ds,eax
66
67     mov ebx,message_1
68     call far [fs:PrintString]
69
70     mov eax,100 ;逻辑扇区号100
71     mov ebx,buffer ;缓冲区偏移地址
72     call far [fs:ReadDiskData] ;段间调用
73
74     mov ebx,message_2
75     call far [fs:PrintString]
76
77     mov ebx,buffer
78     call far [fs:PrintString] ;too.
79
80     jmp far [fs:TerminateProgram] ;将控制权返回到系统
81
82 code_end:
83
84 ;=====
85 SECTION trail
86 ;-----
87 program_end:
```





```

1      ;代码清单13-2
2      ;文件名: c13_core.asm
3      ;文件说明: 保护模式微型核心程序
4      ;创建日期: 2011-10-26 12:11
5
6      ;以下常量定义部分。内核的大部分内容都应当固定
7      core_code_seg_sel    equ    0x38    ;内核代码段选择子
8      core_data_seg_sel    equ    0x30    ;内核数据段选择子
9      sys_routine_seg_sel  equ    0x28    ;系统公共例程代码段的选择子
10     video_ram_seg_sel    equ    0x20    ;视频显示缓冲区的段选择子
11     core_stack_seg_sel   equ    0x18    ;内核堆栈段选择子
12     mem_0_4_gb_seg_sel   equ    0x08    ;整个0-4GB内存的段的选择子
13
14     ;-----
15     ;以下是系统核心的头部,用于加载核心程序
16     core_length          dd core_end    ;核心程序总长度#00
17
18     sys_routine_seg      dd section.sys_routine.start
19                           ;系统公用例程段位置#04
20
21     core_data_seg        dd section.core_data.start
22                           ;核心数据段位置#08
23
24     core_code_seg        dd section.core_code.start
25                           ;核心代码段位置#0c
26
27
28     core_entry            dd start        ;核心代码段入口点#10
29                           dw core_code_seg_sel
30
31     ;=====
32     [bits 32]
33     ;=====
34     SECTION sys_routine vstart=0          ;系统公共例程代码段
35     ;-----
36     ;字符串显示例程
37     put_string:                          ;显示0终止的字符串并移动光标
38                                           ;输入: DS:EBX=串地址
39     push ecx
40     .getc:
41     mov cl,[ebx]
42     or cl,cl
43     jz .exit
44     call put_char
45     inc ebx
46     jmp .getc
47
48     .exit:
49     pop ecx
50     retf                                ;段间返回
51
52     ;-----
53     put_char:                            ;在当前光标处显示一个字符,并推进
54                                           ;光标。仅用于段内调用
55                                           ;输入: CL=字符ASCII码
56     pushad
57
58     ;以下取当前光标位置
59     mov dx,0x3d4
60     mov al,0x0e
61     out dx,al
62     inc dx                                ;0x3d5
63     in al,dx                             ;高字
64     mov ah,al
65
66     dec dx                                ;0x3d4
67     mov al,0x0f
68     out dx,al
69     inc dx                                ;0x3d5
70     in al,dx                             ;低字
71     mov bx,ax                             ;BX=代表光标位置的16位数
72
73     cmp cl,0x0d                          ;回车符?
74     jnz .put_0a
75     mov ax,bx
76     mov bl,80
77     div bl
78     mul bl
79     mov bx,ax
80     jmp .set_cursor
81
82     .put_0a:
83     cmp cl,0x0a                          ;换行符?
84     jnz .put_other
85     add bx,80
86     jmp .roll_screen
87
88     .put_other:                          ;正常显示字符
89     push es
90     mov eax,video_ram_seg_sel            ;0xb8000段的选择子
91     mov es,eax
92     shl bx,1

```

```
93      mov [es:bx],cl
94      pop es
95
96      ;以下将光标位置推进一个字符
97      shr bx,1
98      inc bx
99
100     .roll_screen:
101         cmp bx,2000                ;光标超出屏幕? 滚屏
102         jl .set_cursor
103
104         push ds
105         push es
106         mov eax,video_ram_seg_sel
107         mov ds,eax
108         mov es,eax
109         cld
110         mov esi,0xa0                ;小心! 32位模式下movsb/w/d
111         mov edi,0x00                ;使用的是esi/edi/ecx
112         mov ecx,1920
113         rep movsd
114         mov bx,3840                ;清除屏幕最底一行
115         mov ecx,80                  ;32位程序应该使用ECX
116     .cls:
117         mov word[es:bx],0x0720
118         add bx,2
119         loop .cls
120
121         pop es
122         pop ds
123
124         mov bx,1920
125
126     .set_cursor:
127         mov dx,0x3d4
128         mov al,0x0e
129         out dx,al
130         inc dx                        ;0x3d5
131         mov al,bh
132         out dx,al
133         dec dx                        ;0x3d4
134         mov al,0x0f
135         out dx,al
136         inc dx                        ;0x3d5
137         mov al,bl
138         out dx,al
139
140         popad
141         ret
142
143     ;-----
144     read_hard_disk_0:                ;从硬盘读取一个逻辑扇区
145                                         ;EAX=逻辑扇区号
146                                         ;DS:EBX=目标缓冲地址
147                                         ;返回: EBX=EBX+512
148
149         push eax
150         push ecx
151         push edx
152
153         push eax
154
155         mov dx,0x1f2
156         mov al,1
157         out dx,al                    ;读取的扇区数
158
159         inc dx                        ;0x1f3
160         pop eax
161         out dx,al                    ;LBA地址7~0
162
163         inc dx                        ;0x1f4
164         mov cl,8
165         shr eax,cl
166         out dx,al                    ;LBA地址15~8
167
168         inc dx                        ;0x1f5
169         shr eax,cl
170         out dx,al                    ;LBA地址23~16
171
172         inc dx                        ;0x1f6
173         shr eax,cl
174         or al,0xe0                    ;第一硬盘 LBA地址27~24
175         out dx,al
176
177         inc dx                        ;0x1f7
178         mov al,0x20                    ;读命令
179         out dx,al
180
181     .waits:
182         in al,dx
183         and al,0x88
184         cmp al,0x08
185         jnz .waits                    ;不忙, 且硬盘已准备好数据传输
```

```

185
186         mov ecx,256                ;总共要读取的字数
187         mov dx,0x1f0
188     .readw:
189         in ax,dx
190         mov [ebx],ax
191         add ebx,2
192         loop .readw
193
194         pop edx
195         pop ecx
196         pop eax
197
198         retf                        ;段间返回
199
200 ;-----
201 ;汇编语言程序是极难一次成功，而且调试非常困难。这个例程可以提供帮助
202 put_hex_dword:                    ;在当前光标处以十六进制形式显示
203                                 ;一个双字并推进光标
204                                 ;输入：EDX=要转换并显示的数字
205                                 ;输出：无
206         pushad
207         push ds
208
209         mov ax,core_data_seg_sel    ;切换到核心数据段
210         mov ds,ax
211
212         mov ebx,bin_hex             ;指向核心数据段内的转换表
213         mov ecx,8
214     .xlt:
215         rol edx,4
216         mov eax,edx
217         and eax,0x0000000f
218         xlat
219
220         push ecx
221         mov cl,al
222         call put_char
223         pop ecx
224
225         loop .xlt
226
227         pop ds
228         popad
229         retf
230
231 ;-----
232 allocate_memory:                  ;分配内存
233                                 ;输入：ECX=希望分配的字节数
234                                 ;输出：ECX=起始线性地址
235         push ds
236         push eax
237         push ebx
238
239         mov eax,core_data_seg_sel
240         mov ds,eax
241
242         mov eax,[ram_alloc]
243         add eax,ecx                 ;下一次分配时的起始地址
244
245         ;这里应当有检测可用内存数量的指令
246
247         mov ecx,[ram_alloc]        ;返回分配的起始地址
248
249         mov ebx,eax
250         and ebx,0xffffffffc
251         add ebx,4                   ;强制对齐
252         test eax,0x00000003         ;下次分配的起始地址最好是4字节对齐
253         cmovnz eax,ebx             ;如果没有对齐，则强制对齐
254         mov [ram_alloc],eax        ;下次从该地址分配内存
255                                 ;cmovcc指令可以避免控制转移
256         pop ebx
257         pop eax
258         pop ds
259
260         retf
261
262 ;-----
263 set_up_gdt_descriptor:            ;在GDT内安装一个新的描述符
264                                 ;输入：EDX:EAX=描述符
265                                 ;输出：CX=描述符的选择子
266         push eax
267         push ebx
268         push edx
269
270         push ds
271         push es
272
273         mov ebx,core_data_seg_sel    ;切换到核心数据段
274         mov ds,ebx
275
276         sgdt [pgdt]                ;以便开始处理GDT

```

```

277
278     mov ebx,mem_0_4_gb_seg_sel
279     mov es,ebx
280
281     movzx ebx,word [pgdt]           ;GDT界限
282     inc bx                          ;GDT总字节数，也是下一个描述符偏移
283     add ebx,[pgdt+2]               ;下一个描述符的线性地址
284
285     mov [es:ebx],eax
286     mov [es:ebx+4],edx
287
288     add word [pgdt],8              ;增加一个描述符的大小
289
290     lgdt [pgdt]                    ;对GDT的更改生效
291
292     mov ax,[pgdt]                  ;得到GDT界限值
293     xor dx,dx
294     mov bx,8
295     div bx                         ;除以8，去掉余数
296     mov cx,ax
297     shl cx,3                       ;将索引号移到正确位置
298
299     pop es
300     pop ds
301
302     pop edx
303     pop ebx
304     pop eax
305
306     retf
307 ;-----
308 make_seg_descriptor:               ;构造存储器和系统的段描述符
309                                     ;输入： EAX=线性基地址
310                                     ;       EBX=段界限
311                                     ;       ECX=属性。各属性位都在原始
312                                     ;       位置，无关的位清零
313                                     ;返回： EDX:EAX=描述符
314
315     mov edx,eax
316     shl eax,16
317     or ax,bx                       ;描述符前32位(EAX)构造完毕
318
319     and edx,0xffff0000             ;清除基地址中无关的位
320     rol edx,8
321     bswap edx                      ;装配基址的31~24和23~16 (80486+)
322
323     xor bx,bx
324     or edx,ebx                     ;装配段界限的高4位
325
326     or edx,ecx                     ;装配属性
327
328     retf
329 ;=====
330 SECTION core_data vstart=0         ;系统核心的数据段
331 ;-----
332     pgdt                dw 0          ;用于设置和修改GDT
333                        dd 0
334
335     ram_alloc            dd 0x00100000 ;下次分配内存时的起始地址
336
337 ;符号地址检索表
338 salt:
339 salt_1                  db '@PrintString'
340                        times 256-($-salt_1) db 0
341                        dd put_string
342                        dw sys_routine_seg_sel
343
344 salt_2                  db '@ReadDiskData'
345                        times 256-($-salt_2) db 0
346                        dd read_hard_disk_0
347                        dw sys_routine_seg_sel
348
349 salt_3                  db '@PrintDwordAsHexString'
350                        times 256-($-salt_3) db 0
351                        dd put_hex_dword
352                        dw sys_routine_seg_sel
353
354 salt_4                  db '@TerminateProgram'
355                        times 256-($-salt_4) db 0
356                        dd return_point
357                        dw core_code_seg_sel
358
359 salt_item_len            equ $-salt_4
360 salt_items               equ ($-salt)/salt_item_len
361
362 message_1               db ' If you seen this message,that means we '
363                        db 'are now in protect mode,and the system '
364                        db 'core is loaded,and the video display '
365                        db 'routine works perfectly.',0x0d,0x0a,0
366
367 message_5               db ' Loading user program...',0
368

```

```

369         do_status      db  'Done.',0x0d,0x0a,0
370
371         message_6       db  0x0d,0x0a,0x0d,0x0a,0x0d,0x0a
372                        db  ' User program terminated,control returned.',0
373
374         bin_hex         db  '0123456789ABCDEF'
375                        ;put_hex_dword子过程用的查找表
376         core_buf    times 2048 db 0                ;内核用的缓冲区
377
378         esp_pointer     dd  0                      ;内核用来临时保存自己的栈指针
379
380         cpu_brnd0       db  0x0d,0x0a,' ',0
381         cpu_brand    times 52 db 0
382         cpu_brnd1      db  0x0d,0x0a,0x0d,0x0a,0
383
384 ;=====
385 SECTION core_code vstart=0
386 ;-----
387 load_relocate_program:                ;加载并重定位用户程序
388                                     ;输入: ESI=起始逻辑扇区号
389                                     ;返回: AX=指向用户程序头部的选择子
390
391         push ebx
392         push ecx
393         push edx
394         push esi
395         push edi
396
397         push ds
398         push es
399
400         mov eax,core_data_seg_sel
401         mov ds,eax                    ;切换DS到内核数据段
402
403         mov eax,esi                    ;读取程序头部数据
404         mov ebx,core_buf
405         call sys_routine_seg_sel:read_hard_disk_0
406
407 ;以下判断整个程序有多大
408 mov eax,[core_buf]                    ;程序尺寸
409 mov ebx,eax
410 and ebx,0xfffffe00                    ;使之512字节对齐 (能被512整除的数,
411 add ebx,512                            ;低9位都为0
412 test eax,0x000001ff                    ;程序的大小正好是512的倍数吗?
413 cmovnz eax,ebx                        ;不是。使用凑整的结果
414
415 mov ecx,eax                            ;实际需要申请的内存数量
416 call sys_routine_seg_sel:allocate_memory
417 mov ebx,ecx                            ;ebx -> 申请到的内存首地址
418 push ebx                                ;保存该首地址
419 xor edx,edx
420 mov ecx,512
421 div ecx
422 mov ecx,eax                            ;总扇区数
423
424 mov eax,mem_0_4_gb_seg_sel            ;切换DS到0-4GB的段
425 mov ds,eax
426
427 mov eax,esi                            ;起始扇区号
428 .b1:
429 call sys_routine_seg_sel:read_hard_disk_0
430 inc eax
431 loop .b1                                ;循环读,直到读完整个用户程序
432
433 ;建立程序头部段描述符
434 pop edi                                ;恢复程序装载的首地址
435 mov eax,edi                            ;程序头部起始线性地址
436 mov ebx,[edi+0x04]                    ;段长度
437 dec ebx                                ;段界限
438 mov ecx,0x00409200                    ;字节粒度的数据段描述符
439 call sys_routine_seg_sel:make_seg_descriptor
440 call sys_routine_seg_sel:set_up_gdt_descriptor
441 mov [edi+0x04],cx
442
443 ;建立程序代码段描述符
444 mov eax,edi
445 add eax,[edi+0x14]                    ;代码起始线性地址
446 mov ebx,[edi+0x18]                    ;段长度
447 dec ebx                                ;段界限
448 mov ecx,0x00409800                    ;字节粒度的代码段描述符
449 call sys_routine_seg_sel:make_seg_descriptor
450 call sys_routine_seg_sel:set_up_gdt_descriptor
451 mov [edi+0x14],cx
452
453 ;建立程序数据段描述符
454 mov eax,edi
455 add eax,[edi+0x1c]                    ;数据段起始线性地址
456 mov ebx,[edi+0x20]                    ;段长度
457 dec ebx                                ;段界限
458 mov ecx,0x00409200                    ;字节粒度的数据段描述符
459 call sys_routine_seg_sel:make_seg_descriptor
460 call sys_routine_seg_sel:set_up_gdt_descriptor
461 mov [edi+0x1c],cx

```

```

461         ;建立程序堆栈段描述符
462         mov ecx,[edi+0x0c]                ;4KB的倍率
463         mov ebx,0x000fffff
464         sub ebx,ecx                        ;得到段界限
465         mov eax,4096
466         mul dword [edi+0x0c]
467         mov ecx,eax                        ;准备为堆栈分配内存
468         call sys_routine_seg_sel:allocate_memory
469         add eax,ecx                        ;得到堆栈的高端物理地址
470         mov ecx,0x00c09600                ;4KB粒度的堆栈段描述符
471         call sys_routine_seg_sel:make_seg_descriptor
472         call sys_routine_seg_sel:set_up_gdt_descriptor
473         mov [edi+0x08],cx
474
475         ;重定位SALT
476         mov eax,[edi+0x04]
477         mov es,eax                        ;es -> 用户程序头部
478         mov eax,core_data_seg_sel
479         mov ds,eax
480
481         cld
482
483         mov ecx,[es:0x24]                  ;用户程序的SALT条目数
484         mov edi,0x28                      ;用户程序内的SALT位于头部内0x2c处
485     .b2:
486         push ecx
487         push edi
488
489         mov ecx,salt_items
490         mov esi,salt
491     .b3:
492         push edi
493         push esi
494         push ecx
495
496         mov ecx,64                        ;检索表中，每条目的比较次数
497         repe cmpsd                         ;每次比较4字节
498         jnz .b4
499         mov eax,[esi]
500         mov [es:edi-256],eax              ;若匹配，esi恰好指向其后的地址数据
501         mov ax,[esi+4]                    ;将字符串改写成偏移地址
502         mov [es:edi-252],ax
503         ;以及段选择子
504     .b4:
505         pop ecx
506         pop esi
507         add esi,salt_item_len
508         pop edi
509         ;从头比较
510         loop .b3
511
512         pop edi
513         add edi,256
514         pop ecx
515         loop .b2
516
517         mov ax,[es:0x04]
518
519         pop es                            ;恢复到调用此过程前的es段
520         pop ds                            ;恢复到调用此过程前的ds段
521
522         pop edi
523         pop esi
524         pop edx
525         pop ecx
526         pop ebx
527
528         ret
529
530 ;-----
531 start:
532     mov ecx,core_data_seg_sel              ;使ds指向核心数据段
533     mov ds,ecx
534
535     mov ebx,message_1
536     call sys_routine_seg_sel:put_string
537
538     ;显示处理器品牌信息
539     mov eax,0x80000002
540     cpuid
541     mov [cpu_brand + 0x00],eax
542     mov [cpu_brand + 0x04],ebx
543     mov [cpu_brand + 0x08],ecx
544     mov [cpu_brand + 0x0c],edx
545
546     mov eax,0x80000003
547     cpuid
548     mov [cpu_brand + 0x10],eax
549     mov [cpu_brand + 0x14],ebx
550     mov [cpu_brand + 0x18],ecx
551     mov [cpu_brand + 0x1c],edx
552

```

```
553      mov eax,0x80000004
554      cpuid
555      mov [cpu_brand + 0x20],eax
556      mov [cpu_brand + 0x24],ebx
557      mov [cpu_brand + 0x28],ecx
558      mov [cpu_brand + 0x2c],edx
559
560      mov ebx,cpu_brnd0
561      call sys_routine_seg_sel:put_string
562      mov ebx,cpu_brand
563      call sys_routine_seg_sel:put_string
564      mov ebx,cpu_brnd1
565      call sys_routine_seg_sel:put_string
566
567      mov ebx,message_5
568      call sys_routine_seg_sel:put_string
569      mov esi,50                      ;用户程序位于逻辑50扇区
570      call load_relocate_program
571
572      mov ebx,do_status
573      call sys_routine_seg_sel:put_string
574
575      mov [esp_pointer],esp          ;临时保存堆栈指针
576
577      mov ds,ax
578
579      jmp far [0x10]                 ;控制权交给用户程序（入口点）
580                                      ;堆栈可能切换
581
582      return_point:                  ;用户程序返回点
583      mov eax,core_data_seg_sel      ;使ds指向核心数据段
584      mov ds,eax
585
586      mov eax,core_stack_seg_sel      ;切换回内核自己的堆栈
587      mov ss,eax
588      mov esp,[esp_pointer]
589
590      mov ebx,message_6
591      call sys_routine_seg_sel:put_string
592
593      ;这里可以放置清除用户程序各种描述符的指令
594      ;也可以加载并启动其它程序
595
596      hlt
597
598      ;=====
599      SECTION core_trail
600      ;-----
601      core_end:
```





```

1      ;代码清单13-1
2      ;文件名: c13_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码
4      ;创建日期: 2011-10-28 22:35          ;设置堆栈段和栈指针
5
6      core_base_address equ 0x00040000      ;常数, 内核加载的起始内存地址
7      core_start_sector equ 0x00000001      ;常数, 内核的起始逻辑扇区号
8
9      mov ax,cs
10     mov ss,ax
11     mov sp,0x7c00
12
13     ;计算GDT所在的逻辑段地址
14     mov eax,[cs:pgdt+0x7c00+0x02]          ;GDT的32位物理地址
15     xor edx,edx
16     mov ebx,16
17     div ebx                                ;分解成16位逻辑地址
18
19     mov ds,eax
20     mov ebx,edx                            ;令DS指向该段以进行操作
21                                           ;段内起始偏移地址
22
23     ;跳过0#号描述符的槽位
24     ;创建1#描述符, 这是一个数据段, 对应0~4GB的线性地址空间
25     mov dword [ebx+0x08],0x0000ffff        ;基地址为0, 段界限为0xFFFFF
26     mov dword [ebx+0x0c],0x00cf9200        ;粒度为4KB, 存储器段描述符
27
28     ;创建保护模式下初始代码段描述符
29     mov dword [ebx+0x10],0x7c0001ff        ;基地址为0x00007c00, 界限0x1FF
30     mov dword [ebx+0x14],0x00409800        ;粒度为1个字节, 代码段描述符
31
32     ;建立保护模式下的堆栈段描述符
33     mov dword [ebx+0x18],0x7c00fffe        ;基地址为0x00007c00, 界限0xFFFFE
34     mov dword [ebx+0x1c],0x00cf9600        ;粒度为4KB
35
36     ;建立保护模式下的显示缓冲区描述符
37     mov dword [ebx+0x20],0x80007fff        ;基地址为0x000B8000, 界限0x07FFF
38     mov dword [ebx+0x24],0x0040920b        ;粒度为字节
39
40     ;初始化描述符表寄存器GDTR
41     mov word [cs: pgdt+0x7c00],39          ;描述符表的界限
42
43     lgdt [cs: pgdt+0x7c00]
44
45     in al,0x92                             ;南桥芯片内的端口
46     or al,0000_0010B
47     out 0x92,al                            ;打开A20
48
49     cli                                    ;中断机制尚未工作
50
51     mov eax,cr0
52     or eax,1
53     mov cr0,eax                            ;设置PE位
54
55     ;以下进入保护模式... ...
56     jmp dword 0x0010:flush
57
58     [bits 32]
59     flush: mov eax,0x0008                   ;加载数据段(0..4GB)选择子
60     mov ds,eax
61
62     mov eax,0x0018
63     mov ss,eax
64     xor esp,esp                            ;堆栈指针 <- 0
65
66     ;以下加载系统核心程序
67     mov edi,core_base_address
68
69     mov eax,core_start_sector
70     mov ebx,edi                            ;起始地址
71     call read_hard_disk_0                  ;以下读取程序的起始部分 (一个扇区)
72
73     ;以下判断整个程序有多大
74     mov eax,[edi]                          ;核心程序尺寸
75     xor edx,edx
76     mov ecx,512                            ;512字节每扇区
77     div ecx
78
79     or edx,edx
80     jnz @1
81     dec eax
82
83     @1: or eax,eax                         ;考虑实际长度≤512个字节的情况
84     jz setup                              ;EAX=0 ?
85
86     ;读取剩余的扇区
87     mov ecx,eax                            ;32位模式下的LOOP使用ECX
88     mov eax,core_start_sector
89     inc eax                                ;从下一个逻辑扇区接着读
90
91     @2: call read_hard_disk_0
92     inc eax

```

```

93         loop @2                                ;循环读，直到读完整个内核
94
95     setup:
96         mov esi,[0x7c00+pgdt+0x02]              ;不可以在代码段内寻址pgdt，但可以
97                                                 ;通过4GB的段来访问
98         ;建立公用例程段描述符
99         mov eax,[edi+0x04]                      ;公用例程代码段起始汇编地址
100        mov ebx,[edi+0x08]                      ;核心数据段汇编地址
101        sub ebx,eax
102        dec ebx                                ;公用例程段界限
103        add eax,edi                            ;公用例程段基地址
104        mov ecx,0x00409800                    ;字节粒度的代码段描述符
105        call make_gdt_descriptor
106        mov [esi+0x28],eax
107        mov [esi+0x2c],edx
108
109        ;建立核心数据段描述符
110        mov eax,[edi+0x08]                    ;核心数据段起始汇编地址
111        mov ebx,[edi+0x0c]                    ;核心代码段汇编地址
112        sub ebx,eax
113        dec ebx                                ;核心数据段界限
114        add eax,edi                            ;核心数据段基地址
115        mov ecx,0x00409200                    ;字节粒度的数据段描述符
116        call make_gdt_descriptor
117        mov [esi+0x30],eax
118        mov [esi+0x34],edx
119
120        ;建立核心代码段描述符
121        mov eax,[edi+0x0c]                    ;核心代码段起始汇编地址
122        mov ebx,[edi+0x00]                    ;程序总长度
123        sub ebx,eax
124        dec ebx                                ;核心代码段界限
125        add eax,edi                            ;核心代码段基地址
126        mov ecx,0x00409800                    ;字节粒度的代码段描述符
127        call make_gdt_descriptor
128        mov [esi+0x38],eax
129        mov [esi+0x3c],edx
130
131        mov word [0x7c00+pgdt],63            ;描述符表的界限
132
133        lgdt [0x7c00+pgdt]
134
135        jmp far [edi+0x10]
136
137 ;-----
138 read_hard_disk_0:                            ;从硬盘读取一个逻辑扇区
139                                                 ;EAX=逻辑扇区号
140                                                 ;DS:EBX=目标缓冲区地址
141                                                 ;返回: EBX=EBX+512
142
143         push eax
144         push ecx
145         push edx
146
147         push eax
148
149         mov dx,0x1f2
150         mov al,1
151         out dx,al                            ;读取的扇区数
152
153         inc dx                                ;0x1f3
154         pop eax
155         out dx,al                            ;LBA地址7~0
156
157         inc dx                                ;0x1f4
158         mov cl,8
159         shr eax,cl
160         out dx,al                            ;LBA地址15~8
161
162         inc dx                                ;0x1f5
163         shr eax,cl
164         out dx,al                            ;LBA地址23~16
165
166         inc dx                                ;0x1f6
167         shr eax,cl
168         or al,0xe0
169         out dx,al                            ;第一硬盘 LBA地址27~24
170
171         inc dx                                ;0x1f7
172         mov al,0x20
173         out dx,al                            ;读命令
174
175     .waits:
176         in al,dx
177         and al,0x88
178         cmp al,0x08
179         jnz .waits                            ;不忙，且硬盘已准备好数据传输
180
181         mov ecx,256
182         mov dx,0x1f0
183     .readw:
184         in ax,dx
185         mov [ebx],ax

```

```
185         add ebx,2
186         loop .readw
187
188         pop edx
189         pop ecx
190         pop eax
191
192         ret
193
194 ;-----
195 make_gdt_descriptor:           ;构造描述符
196                               ;输入: EAX=线性基地址
197                               ;      EBX=段界限
198                               ;      ECX=属性 (各属性位都在原始
199                               ;      位置, 其它没用到的位置0)
200                               ;返回: EDX:EAX=完整的描述符
201
202         mov edx,eax
203         shl eax,16
204         or ax,bx               ;描述符前32位(EAX)构造完毕
205
206         and edx,0xffff0000    ;清除基地址中无关的位
207         rol edx,8
208         bswap edx             ;装配基址的31~24和23~16 (80486+)
209
210         xor bx,bx
211         or edx,ebx            ;装配段界限的高4位
212
213         or edx,ecx            ;装配属性
214
215         ret
216 ;-----
217         pgdt                 dw 0
218                               dd 0x00007e00 ;GDT的物理地址
219 ;-----
220         times 510-($-$$) db 0
221                               db 0x55,0xaa
```



```

1      ;代码清单14-1
2      ;文件名: c14_core.asm
3      ;文件说明: 保护模式微型核心程序
4      ;创建日期: 2011-11-6 18:37
5
6      ;以下常量定义部分。内核的大部分内容都应当固定
7      core_code_seg_sel    equ    0x38    ;内核代码段选择子
8      core_data_seg_sel    equ    0x30    ;内核数据段选择子
9      sys_routine_seg_sel   equ    0x28    ;系统公共例程代码段的选择子
10     video_ram_seg_sel     equ    0x20    ;视频显示缓冲区的段选择子
11     core_stack_seg_sel    equ    0x18    ;内核堆栈段选择子
12     mem_0_4_gb_seg_sel    equ    0x08    ;整个0-4GB内存的段的选择子
13
14     ;-----
15     ;以下是系统核心的头部, 用于加载核心程序
16     core_length          dd core_end      ;核心程序总长度#00
17
18     sys_routine_seg      dd section.sys_routine.start
19                           ;系统公用例程段位置#04
20
21     core_data_seg        dd section.core_data.start
22                           ;核心数据段位置#08
23
24     core_code_seg        dd section.core_code.start
25                           ;核心代码段位置#0c
26
27
28     core_entry           dd start          ;核心代码段入口点#10
29                           dw core_code_seg_sel
30
31     ;=====
32     [bits 32]
33     ;=====
34     SECTION sys_routine vstart=0          ;系统公共例程代码段
35     ;-----
36     ;字符串显示例程
37     put_string:                          ;显示0终止的字符串并移动光标
38                                           ;输入: DS:EBX=串地址
39     push ecx
40     .getc:
41     mov cl,[ebx]
42     or cl,cl
43     jz .exit
44     call put_char
45     inc ebx
46     jmp .getc
47
48     .exit:
49     pop ecx
50     retf                                ;段间返回
51
52     ;-----
53     put_char:                            ;在当前光标处显示一个字符, 并推进
54                                           ;光标。仅用于段内调用
55                                           ;输入: CL=字符ASCII码
56     pushad
57
58     ;以下取当前光标位置
59     mov dx,0x3d4
60     mov al,0x0e
61     out dx,al
62     inc dx                                ;0x3d5
63     in al,dx                             ;高字
64     mov ah,al
65
66     dec dx                                ;0x3d4
67     mov al,0x0f
68     out dx,al
69     inc dx                                ;0x3d5
70     in al,dx                             ;低字
71     mov bx,ax                            ;BX=代表光标位置的16位数
72
73     cmp cl,0x0d                          ;回车符?
74     jnz .put_0a
75     mov ax,bx
76     mov bl,80
77     div bl
78     mul bl
79     mov bx,ax
80     jmp .set_cursor
81
82     .put_0a:
83     cmp cl,0x0a                          ;换行符?
84     jnz .put_other
85     add bx,80
86     jmp .roll_screen
87
88     .put_other:                          ;正常显示字符
89     push es
90     mov eax,video_ram_seg_sel            ;0xb8000段的选择子
91     mov es,eax
92     shl bx,1

```

```

93      mov [es:bx],cl
94      pop es
95
96      ;以下将光标位置推进一个字符
97      shr bx,1
98      inc bx
99
100     .roll_screen:
101         cmp bx,2000                ;光标超出屏幕? 滚屏
102         jl .set_cursor
103
104         push ds
105         push es
106         mov eax,video_ram_seg_sel
107         mov ds,eax
108         mov es,eax
109         cld
110         mov esi,0xa0                ;小心! 32位模式下movsb/w/d
111         mov edi,0x00                ;使用的是esi/edi/ecx
112         mov ecx,1920
113         rep movsd
114         mov bx,3840                ;清除屏幕最底一行
115         mov ecx,80                 ;32位程序应该使用ECX
116     .cls:
117         mov word[es:bx],0x0720
118         add bx,2
119         loop .cls
120
121         pop es
122         pop ds
123
124         mov bx,1920
125
126     .set_cursor:
127         mov dx,0x3d4
128         mov al,0x0e
129         out dx,al
130         inc dx                        ;0x3d5
131         mov al,bh
132         out dx,al
133         dec dx                        ;0x3d4
134         mov al,0x0f
135         out dx,al
136         inc dx                        ;0x3d5
137         mov al,bl
138         out dx,al
139
140         popad
141
142         ret
143
144 ;-----
145 read_hard_disk_0:                ;从硬盘读取一个逻辑扇区
146                                ;EAX=逻辑扇区号
147                                ;DS:EBX=目标缓冲区地址
148                                ;返回: EBX=EBX+512
149
150         push eax
151         push ecx
152         push edx
153
154         push eax
155
156         mov dx,0x1f2
157         mov al,1
158         out dx,al                ;读取的扇区数
159
160         inc dx                    ;0x1f3
161         pop eax
162         out dx,al                ;LBA地址7~0
163
164         inc dx                    ;0x1f4
165         mov cl,8
166         shr eax,cl
167         out dx,al                ;LBA地址15~8
168
169         inc dx                    ;0x1f5
170         shr eax,cl
171         out dx,al                ;LBA地址23~16
172
173         inc dx                    ;0x1f6
174         shr eax,cl
175         or al,0xe0                ;第一硬盘 LBA地址27~24
176         out dx,al
177
178         inc dx                    ;0x1f7
179         mov al,0x20                ;读命令
180         out dx,al
181
182     .waits:
183         in al,dx
184         and al,0x88
185         cmp al,0x08

```

```

185         jnz .waits                ;不忙，且硬盘已准备好数据传输
186
187         mov ecx,256                ;总共要读取的字数
188         mov dx,0x1f0
189     .readw:
190         in ax,dx
191         mov [ebx],ax
192         add ebx,2
193         loop .readw
194
195         pop edx
196         pop ecx
197         pop eax
198
199         retf                        ;段间返回
200
201 ;-----
202 ;汇编语言程序是极难一次成功，而且调试非常困难。这个例程可以提供帮助
203 put_hex_dword:                    ;在当前光标处以十六进制形式显示
204                                   ;一个双字并推进光标
205                                   ;输入：EDX=要转换并显示的数字
206                                   ;输出：无
207         pushad
208         push ds
209
210         mov ax,core_data_seg_sel    ;切换到核心数据段
211         mov ds,ax
212
213         mov ebx,bin_hex             ;指向核心数据段内的转换表
214         mov ecx,8
215     .xlt:
216         rol edx,4
217         mov eax,edx
218         and eax,0x0000000f
219         xlat
220
221         push ecx
222         mov cl,al
223         call put_char
224         pop ecx
225
226         loop .xlt
227
228         pop ds
229         popad
230         retf
231
232 ;-----
233 allocate_memory:                  ;分配内存
234                                   ;输入：ECX=希望分配的字节数
235                                   ;输出：ECX=起始线性地址
236         push ds
237         push eax
238         push ebx
239
240         mov eax,core_data_seg_sel
241         mov ds,eax
242
243         mov eax,[ram_alloc]
244         add eax,ecx                 ;下一次分配时的起始地址
245
246         ;这里应当有检测可用内存数量的指令
247
248         mov ecx,[ram_alloc]        ;返回分配的起始地址
249
250         mov ebx,eax
251         and ebx,0xffffffffc
252         add ebx,4                   ;强制对齐
253         test eax,0x00000003         ;下次分配的起始地址最好是4字节对齐
254         cmovnz eax,ebx             ;如果没有对齐，则强制对齐
255         mov [ram_alloc],eax        ;下次从该地址分配内存
256                                   ;cmovcc指令可以避免控制转移
257         pop ebx
258         pop eax
259         pop ds
260
261         retf
262
263 ;-----
264 set_up_gdt_descriptor:            ;在GDT内安装一个新的描述符
265                                   ;输入：EDX:EAX=描述符
266                                   ;输出：CX=描述符的选择子
267         push eax
268         push ebx
269         push edx
270
271         push ds
272         push es
273
274         mov ebx,core_data_seg_sel  ;切换到核心数据段
275         mov ds,ebx
276

```

```

277         sgdt [pgdt]                ;以便开始处理GDT
278
279         mov ebx,mem_0_4_gb_seg_sel
280         mov es,ebx
281
282         movzx ebx,word [pgdt]        ;GDT界限
283         inc bx                       ;GDT总字节数,也是下一个描述符偏移
284         add ebx,[pgdt+2]             ;下一个描述符的线性地址
285
286         mov [es:ebx],eax
287         mov [es:ebx+4],edx
288
289         add word [pgdt],8            ;增加一个描述符的大小
290
291         lgdt [pgdt]                 ;对GDT的更改生效
292
293         mov ax,[pgdt]               ;得到GDT界限值
294         xor dx,dx
295         mov bx,8
296         div bx                      ;除以8,去掉余数
297         mov cx,ax
298         shl cx,3                    ;将索引号移到正确位置
299
300         pop es
301         pop ds
302
303         pop edx
304         pop ebx
305         pop eax
306
307         retf
308 ;-----
309 make_seg_descriptor:                ;构造存储器和系统的段描述符
310                                     ;输入: EAX=线性基地址
311                                     ;       EBX=段界限
312                                     ;       ECX=属性。各属性位都在原始
313                                     ;       位置,无关的位清零
314                                     ;返回: EDX:EAX=描述符
315         mov edx,eax
316         shl eax,16
317         or ax,bx                    ;描述符前32位(EAX)构造完毕
318
319         and edx,0xffff0000          ;清除基地址中无关的位
320         rol edx,8
321         bswap edx                   ;装配基址的31~24和23~16 (80486+)
322
323         xor bx,bx
324         or edx,ebx                  ;装配段界限的高4位
325
326         or edx,ecx                  ;装配属性
327
328         retf
329 ;-----
330 make_gate_descriptor:               ;构造门的描述符(调用门等)
331                                     ;输入: EAX=门代码在段内偏移地址
332                                     ;       BX=门代码所在段的选择子
333                                     ;       CX=段类型及属性等(各属
334                                     ;       性位都在原始位置)
335                                     ;返回: EDX:EAX=完整的描述符
336
337         push ebx
338         push ecx
339
340         mov edx,eax
341         and edx,0xffff0000          ;得到偏移地址高16位
342         or dx,cx                    ;组装属性部分到EDX
343
344         and eax,0x0000ffff          ;得到偏移地址低16位
345         shl ebx,16
346         or eax,ebx                  ;组装段选择子部分
347
348         pop ecx
349         pop ebx
350
351         retf
352
353 sys_routine_end:
354
355 ;=====
356 SECTION core_data vstart=0          ;系统核心的数据段
357 ;-----
358         pgdt                dw 0      ;用于设置和修改GDT
359                             dd 0
360
361         ram_alloc           dd 0x00100000 ;下次分配内存时的起始地址
362
363         ;符号地址检索表
364         salt:
365         salt_1              db '@PrintString'
366                             times 256-($-salt_1) db 0
367                             dd put_string
368                             dw sys_routine_seg_sel

```



```

369
370     salt_2      db  '@ReadDiskData'
371                times 256-($-salt_2) db 0
372                dd  read_hard_disk_0
373                dw  sys_routine_seg_sel
374
375     salt_3      db  '@PrintDwordAsHexString'
376                times 256-($-salt_3) db 0
377                dd  put_hex_dword
378                dw  sys_routine_seg_sel
379
380     salt_4      db  '@TerminateProgram'
381                times 256-($-salt_4) db 0
382                dd  return_point
383                dw  core_code_seg_sel
384
385     salt_item_len equ $-salt_4
386     salt_items    equ ($-salt)/salt_item_len
387
388     message_1    db  ' If you seen this message,that means we '
389                db  'are now in protect mode,and the system '
390                db  'core is loaded,and the video display '
391                db  'routine works perfectly.',0x0d,0x0a,0
392
393     message_2    db  ' System wide CALL-GATE mounted.',0x0d,0x0a,0
394
395     message_3    db  0x0d,0x0a,' Loading user program...',0
396
397     do_status     db  'Done.',0x0d,0x0a,0
398
399     message_6     db  0x0d,0x0a,0x0d,0x0a,0x0d,0x0a
400                db  ' User program terminated,control returned.',0
401
402     bin_hex       db  '0123456789ABCDEF'
403                ;put_hex_dword子过程用的查找表
404
405     core_buf      times 2048 db 0 ;内核用的缓冲区
406
407     esp_pointer   dd  0 ;内核用来临时保存自己的栈指针
408
409     cpu_brnd0     db  0x0d,0x0a,' ',0
410     cpu_brand     times 52 db 0
411     cpu_brnd1     db  0x0d,0x0a,0x0d,0x0a,0
412
413     ;任务控制块链
414     tcb_chain     dd  0
415
416 core_data_end:
417
418 ;=====
419 SECTION core_code vstart=0
420 ;-----
421 fill_descriptor_in_ldt: ;在LDT内安装一个新的描述符
422                        ;输入: EDX:EAX=描述符
423                        ; EBX=TCB基地址
424                        ;输出: CX=描述符的选择子
425
426     push eax
427     push edx
428     push edi
429     push ds
430
431     mov ecx,mem_0_4_gb_seg_sel
432     mov ds,ecx
433
434     mov edi,[ebx+0x0c] ;获得LDT基地址
435
436     xor ecx,ecx
437     mov cx,[ebx+0x0a] ;获得LDT界限
438     inc cx ;LDT的总字节数,即新描述符偏移地址
439
440     mov [edi+ecx+0x00],eax
441     mov [edi+ecx+0x04],edx ;安装描述符
442
443     add cx,8
444     dec cx ;得到新的LDT界限值
445
446     mov [ebx+0x0a],cx ;更新LDT界限值到TCB
447
448     mov ax,cx
449     xor dx,dx
450     mov cx,8
451     div cx
452
453     mov cx,ax
454     shl cx,3 ;左移3位,并且
455     or cx,0000_0000_0000_0100B ;使TI位=1,指向LDT,最后使RPL=00
456
457     pop ds
458     pop edi
459     pop edx
460     pop eax

```

```

461         ret
462
463 ;-----
464 load_relocate_program:                ;加载并重定位用户程序
465                                         ;输入: PUSH 逻辑扇区号
466                                         ;      PUSH 任务控制块基地址
467                                         ;输出: 无
468
469     pushad
470
471     push ds
472     push es
473
474     mov ebp,esp                        ;为访问通过堆栈传递的参数做准备
475
476     mov ecx,mem_0_4_gb_seg_sel
477     mov es,ecx
478
479     mov esi,[ebp+11*4]                 ;从堆栈中取得TCB的基地址
480
481     ;以下申请创建LDT所需要的内存
482     mov ecx,160                       ;允许安装20个LDT描述符
483     call sys_routine_seg_sel:allocate_memory
484     mov [es:esi+0x0c],ecx              ;登记LDT基地址到TCB中
485     mov word [es:esi+0x0a],0xffff      ;登记LDT初始的界限到TCB中
486
487     ;以下开始加载用户程序
488     mov eax,core_data_seg_sel
489     mov ds,eax                        ;切换DS到内核数据段
490
491     mov eax,[ebp+12*4]                 ;从堆栈中取出用户程序起始扇区号
492     mov ebx,core_buf                  ;读取程序头部数据
493     call sys_routine_seg_sel:read_hard_disk_0
494
495     ;以下判断整个程序有多大
496     mov eax,[core_buf]                 ;程序尺寸
497     mov ebx,eax
498     and ebx,0xfffffe00                 ;使之512字节对齐 (能被512整除的数低
499     add ebx,512                       ;9位都为0
500     test eax,0x000001ff                ;程序的大小正好是512的倍数吗?
501     cmovnz eax,ebx                     ;不是。使用凑整的结果
502
503     mov ecx,eax                        ;实际需要申请的内存数量
504     call sys_routine_seg_sel:allocate_memory
505     mov [es:esi+0x06],ecx              ;登记程序加载基地址到TCB中
506
507     mov ebx,ecx                        ;ebx -> 申请到的内存首地址
508     xor edx,edx
509     mov ecx,512
510     div ecx
511     mov ecx,eax                        ;总扇区数
512
513     mov eax,mem_0_4_gb_seg_sel         ;切换DS到0-4GB的段
514     mov ds,eax
515
516     mov eax,[ebp+12*4]                 ;起始扇区号
517     .b1:
518     call sys_routine_seg_sel:read_hard_disk_0
519     inc eax
520     loop .b1                           ;循环读, 直到读完整个用户程序
521
522     mov edi,[es:esi+0x06]              ;获得程序加载基地址
523
524     ;建立程序头部段描述符
525     mov eax,edi                        ;程序头部起始线性地址
526     mov ebx,[edi+0x04]                 ;段长度
527     dec ebx                             ;段界限
528     mov ecx,0x0040f200                 ;字节粒度的数据段描述符, 特权级3
529     call sys_routine_seg_sel:make_seg_descriptor
530
531     ;安装头部段描述符到LDT中
532     mov ebx,esi                        ;TCB的基地址
533     call fill_descriptor_in_ldt
534
535     or cx,0000_0000_0000_0011B        ;设置选择子的特权级为3
536     mov [es:esi+0x44],cx              ;登记程序头部段选择子到TCB
537     mov [edi+0x04],cx                  ;和头部内
538
539     ;建立程序代码段描述符
540     mov eax,edi                        ;代码起始线性地址
541     add eax,[edi+0x14]                 ;段长度
542     mov ebx,[edi+0x18]                 ;段界限
543     dec ebx                             ;段界限
544     mov ecx,0x0040f800                 ;字节粒度的代码段描述符, 特权级3
545     call sys_routine_seg_sel:make_seg_descriptor
546     mov ebx,esi                        ;TCB的基地址
547     call fill_descriptor_in_ldt
548
549     or cx,0000_0000_0000_0011B        ;设置选择子的特权级为3
550     mov [edi+0x14],cx                  ;登记代码段选择子到头部
551
552     ;建立程序数据段描述符
553     mov eax,edi                        ;数据段起始线性地址
554     add eax,[edi+0x1c]

```

```

553      mov ebx,[edi+0x20]          ;段长度
554      dec ebx                    ;段界限
555      mov ecx,0x0040f200         ;字节粒度的数据段描述符, 特权级3
556      call sys_routine_seg_sel:make_seg_descriptor
557      mov ebx,esi                ;TCB的基地址
558      call fill_descriptor_in_ldt
559      or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
560      mov [edi+0x1c],cx          ;登记数据段选择子到头部
561
562      ;建立程序堆栈段描述符
563      mov ecx,[edi+0x0c]          ;4KB的倍率
564      mov ebx,0x000fffff
565      sub ebx,ecx                ;得到段界限
566      mov eax,4096
567      mul ecx
568      mov ecx,eax                ;准备为堆栈分配内存
569      call sys_routine_seg_sel:allocate_memory
570      add eax,ecx                ;得到堆栈的高端物理地址
571      mov ecx,0x00c0f600         ;字节粒度的堆栈段描述符, 特权级3
572      call sys_routine_seg_sel:make_seg_descriptor
573      mov ebx,esi                ;TCB的基地址
574      call fill_descriptor_in_ldt
575      or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
576      mov [edi+0x08],cx          ;登记堆栈段选择子到头部
577
578      ;重定位SALT
579      mov eax,mem_0_4_gb_seg_sel ;这里和前一章不同, 头部段描述符
580      mov es,eax                 ;已安装, 但还没有生效, 故只能通
581                                   ;过4GB段访问用户程序头部
582      mov eax,core_data_seg_sel
583      mov ds,eax
584
585      cld
586
587      mov ecx,[es:edi+0x24]        ;U-SALT条目数(通过访问4GB段取得)
588      add edi,0x28                ;U-SALT在4GB段内的偏移
589
590      .b2:
591      push ecx
592      push edi
593
594      mov ecx,salt_items
595      mov esi,salt
596
597      .b3:
598      push edi
599      push esi
600      push ecx
601
602      mov ecx,64                  ;检索表中, 每条目的比较次数
603      repe cmpsd                  ;每次比较4字节
604      jnz .b4
605      mov eax,[esi]               ;若匹配, 则esi恰好指向其后的地址
606      mov [es:edi-256],eax        ;将字符串改写成偏移地址
607      mov ax,[esi+4]
608      or ax,0000000000000011B    ;以用户程序自己的特权级使用调用门
609                                   ;故RPL=3
610      mov [es:edi-252],ax         ;回填调用门选择子
611
612      .b4:
613      pop ecx
614      pop esi
615      add esi,salt_item_len
616      pop edi
617      loop .b3
618
619      pop edi
620      add edi,256
621      pop ecx
622      loop .b2
623
624      mov esi,[ebp+11*4]          ;从堆栈中取得TCB的基地址
625
626      ;创建0特权级堆栈
627      mov ecx,4096
628      mov eax,ecx
629      mov [es:esi+0x1a],ecx       ;为生成堆栈高端地址做准备
630      shr dword [es:esi+0x1a],12 ;登记0特权级堆栈尺寸到TCB
631      call sys_routine_seg_sel:allocate_memory
632      add eax,ecx                ;堆栈必须使用高端地址为基地址
633      mov [es:esi+0x1e],eax       ;登记0特权级堆栈基地址到TCB
634      mov ebx,0xfffffe           ;段长度(界限)
635      mov ecx,0x00c09600         ;4KB粒度, 读写, 特权级0
636      call sys_routine_seg_sel:make_seg_descriptor
637      mov ebx,esi                ;TCB的基地址
638      call fill_descriptor_in_ldt
639      ;or cx,0000_0000_0000_0000 ;设置选择子的特权级为0
640      mov [es:esi+0x22],cx        ;登记0特权级堆栈选择子到TCB
641      mov dword [es:esi+0x24],0   ;登记0特权级堆栈初始ESP到TCB
642
643      ;创建1特权级堆栈
644      mov ecx,4096
645      mov eax,ecx
646      mov [es:esi+0x28],ecx       ;为生成堆栈高端地址做准备

```

```

645     shr [es:esi+0x28],12           ;登记1特权级堆栈尺寸到TCB
646     call sys_routine_seg_sel:allocate_memory
647     add eax,ecx                   ;堆栈必须使用高端地址为基地址
648     mov [es:esi+0x2c],eax         ;登记1特权级堆栈基地址到TCB
649     mov ebx,0xfffffe             ;段长度（界限）
650     mov ecx,0x00c0b600           ;4KB粒度，读写，特权级1
651     call sys_routine_seg_sel:make_seg_descriptor
652     mov ebx,esi                   ;TCB的基地址
653     call fill_descriptor_in_ldt
654     or cx,0000_0000_0000_0001    ;设置选择子的特权级为1
655     mov [es:esi+0x30],cx         ;登记1特权级堆栈选择子到TCB
656     mov dword [es:esi+0x32],0    ;登记1特权级堆栈初始ESP到TCB
657
658     ;创建2特权级堆栈
659     mov ecx,4096
660     mov eax,ecx                   ;为生成堆栈高端地址做准备
661     mov [es:esi+0x36],ecx
662     shr [es:esi+0x36],12         ;登记2特权级堆栈尺寸到TCB
663     call sys_routine_seg_sel:allocate_memory
664     add eax,ecx                   ;堆栈必须使用高端地址为基地址
665     mov [es:esi+0x3a],ecx        ;登记2特权级堆栈基地址到TCB
666     mov ebx,0xfffffe             ;段长度（界限）
667     mov ecx,0x00c0d600           ;4KB粒度，读写，特权级2
668     call sys_routine_seg_sel:make_seg_descriptor
669     mov ebx,esi                   ;TCB的基地址
670     call fill_descriptor_in_ldt
671     or cx,0000_0000_0000_0010    ;设置选择子的特权级为2
672     mov [es:esi+0x3e],cx         ;登记2特权级堆栈选择子到TCB
673     mov dword [es:esi+0x40],0    ;登记2特权级堆栈初始ESP到TCB
674
675     ;在GDT中登记LDT描述符
676     mov eax,[es:esi+0x0c]         ;LDT的起始线性地址
677     movzx ebx,word [es:esi+0x0a]  ;LDT段界限
678     mov ecx,0x00408200           ;LDT描述符，特权级0
679     call sys_routine_seg_sel:make_seg_descriptor
680     call sys_routine_seg_sel:set_up_gdt_descriptor
681     mov [es:esi+0x10],cx         ;登记LDT选择子到TCB中
682
683     ;创建用户程序的TSS
684     mov ecx,104                   ;tss的基本尺寸
685     mov [es:esi+0x12],cx
686     dec word [es:esi+0x12]        ;登记TSS界限值到TCB
687     call sys_routine_seg_sel:allocate_memory
688     mov [es:esi+0x14],ecx        ;登记TSS基地址到TCB
689
690     ;登记基本的TSS表格内容
691     mov word [es:ecx+0],0         ;反向链=0
692
693     mov edx,[es:esi+0x24]         ;登记0特权级堆栈初始ESP
694     mov [es:ecx+4],edx           ;到TSS中
695
696     mov dx,[es:esi+0x22]          ;登记0特权级堆栈段选择子
697     mov [es:ecx+8],dx            ;到TSS中
698
699     mov edx,[es:esi+0x32]         ;登记1特权级堆栈初始ESP
700     mov [es:ecx+12],edx          ;到TSS中
701
702     mov dx,[es:esi+0x30]          ;登记1特权级堆栈段选择子
703     mov [es:ecx+16],dx           ;到TSS中
704
705     mov edx,[es:esi+0x40]         ;登记2特权级堆栈初始ESP
706     mov [es:ecx+20],edx          ;到TSS中
707
708     mov dx,[es:esi+0x3e]          ;登记2特权级堆栈段选择子
709     mov [es:ecx+24],dx           ;到TSS中
710
711     mov dx,[es:esi+0x10]          ;登记任务的LDT选择子
712     mov [es:ecx+96],dx           ;到TSS中
713
714     mov dx,[es:esi+0x12]          ;登记任务的I/O位图偏移
715     mov [es:ecx+102],dx          ;到TSS中
716
717     mov word [es:ecx+100],0       ;T=0
718
719     ;在GDT中登记TSS描述符
720     mov eax,[es:esi+0x14]         ;TSS的起始线性地址
721     movzx ebx,word [es:esi+0x12]  ;段长度（界限）
722     mov ecx,0x00408900           ;TSS描述符，特权级0
723     call sys_routine_seg_sel:make_seg_descriptor
724     call sys_routine_seg_sel:set_up_gdt_descriptor
725     mov [es:esi+0x18],cx         ;登记TSS选择子到TCB
726
727     pop es                         ;恢复到调用此过程前的es段
728     pop ds                         ;恢复到调用此过程前的ds段
729
730     popad
731
732     ret 8                          ;丢弃调用本过程前压入的参数
733
734 ;-----
735 append_to_tcb_link:               ;在TCB链上追加任务控制块
736                                  ;输入：ECX=TCB线性基地址

```

```

737         push eax
738         push edx
739         push ds
740         push es
741
742         mov eax,core_data_seg_sel      ;令DS指向内核数据段
743         mov ds,eax
744         mov eax,mem_0_4_gb_seg_sel    ;令ES指向0..4GB段
745         mov es,eax
746
747         mov dword [es: ecx+0x00],0     ;当前TCB指针域清零，以指示这是最
748                                         ;后一个TCB
749
750         mov eax,[tcb_chain]            ;TCB表头指针
751         or eax,eax                     ;链表为空?
752         jz .notcb
753
754     .searc:
755         mov edx,eax
756         mov eax,[es: edx+0x00]
757         or eax,eax
758         jnz .searc
759
760         mov [es: edx+0x00],ecx
761         jmp .retpc
762
763     .notcb:
764         mov [tcb_chain],ecx            ;若为空表，直接令表头指针指向TCB
765
766     .retpc:
767         pop es
768         pop ds
769         pop edx
770         pop eax
771
772         ret
773
774 ;-----
775 start:
776         mov ecx,core_data_seg_sel      ;使ds指向核心数据段
777         mov ds,ecx
778
779         mov ebx,message_1
780         call sys_routine_seg_sel:put_string
781
782         ;显示处理器品牌信息
783         mov eax,0x80000002
784         cpuid
785         mov [cpu_brand + 0x00],eax
786         mov [cpu_brand + 0x04],ebx
787         mov [cpu_brand + 0x08],ecx
788         mov [cpu_brand + 0x0c],edx
789
790         mov eax,0x80000003
791         cpuid
792         mov [cpu_brand + 0x10],eax
793         mov [cpu_brand + 0x14],ebx
794         mov [cpu_brand + 0x18],ecx
795         mov [cpu_brand + 0x1c],edx
796
797         mov eax,0x80000004
798         cpuid
799         mov [cpu_brand + 0x20],eax
800         mov [cpu_brand + 0x24],ebx
801         mov [cpu_brand + 0x28],ecx
802         mov [cpu_brand + 0x2c],edx
803
804         mov ebx,cpu_brnd0              ;显示处理器品牌信息
805         call sys_routine_seg_sel:put_string
806         mov ebx,cpu_brand
807         call sys_routine_seg_sel:put_string
808         mov ebx,cpu_brnd1
809         call sys_routine_seg_sel:put_string
810
811         ;以下开始安装为整个系统服务的调用门。特权级之间的控制转移必须使用门
812         mov edi,salt                    ;C-SALT表的起始位置
813         mov ecx,salt_items              ;C-SALT表的条目数量
814     .b3:
815         push ecx
816         mov eax,[edi+256]                ;该条目入口点的32位偏移地址
817         mov bx,[edi+260]                 ;该条目入口点的段选择子
818         mov cx,1_11_0_1100_000_00000B ;特权级3的调用门(3以上的特权级才
819                                         ;允许访问)，0个参数(因为用寄存器
820                                         ;传递参数，而没有用栈)
821         call sys_routine_seg_sel:make_gate_descriptor
822         call sys_routine_seg_sel:set_up_gdt_descriptor
823         mov [edi+260],cx                 ;将返回的门描述符选择子回填
824         add edi,salt_item_len            ;指向下一个C-SALT条目
825         pop ecx
826         loop .b3
827
828         ;对门进行测试

```

```
829      mov ebx,message_2
830      call far [salt_1+256]          ;通过门显示信息(偏移量将被忽略)
831
832      mov ebx,message_3
833      call sys_routine_seg_sel:put_string ;在内核中调用例程不需要通过门
834
835      ;创建任务控制块。这不是处理器的要求，而是我们自己为了方便而设立的
836      mov ecx,0x46
837      call sys_routine_seg_sel:allocate_memory
838      call append_to_tcb_link        ;将任务控制块追加到TCB链表
839
840      push dword 50                  ;用户程序位于逻辑50扇区
841      push ecx                       ;压入任务控制块起始线性地址
842
843      call load_relocate_program
844
845      mov ebx,do_status
846      call sys_routine_seg_sel:put_string
847
848      mov eax,mem_0_4_gb_seg_sel
849      mov ds,eax
850
851      ltr [ecx+0x18]                ;加载任务状态段
852      lldt [ecx+0x10]               ;加载LDT
853
854      mov eax,[ecx+0x44]
855      mov ds,eax                   ;切换到用户程序头部段
856
857      ;以下假装是从调用门返回。摹仿处理器压入返回参数
858      push dword [0x08]             ;调用前的堆栈段选择子
859      push dword 0                  ;调用前的esp
860
861      push dword [0x14]             ;调用前的代码段选择子
862      push dword [0x10]             ;调用前的eip
863
864      retf
865
866      return_point:                 ;用户程序返回点
867      mov eax,core_data_seg_sel     ;因为c14.asm是以JMP的方式使用调
868      mov ds,eax                   ;用门@TerminateProgram，回到这
869                                   ;里时，特权级为3，会导致异常。
870      mov ebx,message_6
871      call sys_routine_seg_sel:put_string
872
873      hlt
874
875      core_code_end:
876
877      ;-----
878      SECTION core_trail
879      ;-----
880      core_end:
```

```
1 ;代码清单15-2
2 ;文件名: c15.asm
3 ;文件说明: 用户程序
4 ;创建日期: 2011-11-15 19:11
5
6 ;=====
7 SECTION header vstart=0
8
9     program_length dd program_end ;程序总长度#0x00
10
11     head_len dd header_end ;程序头部的长度#0x04
12
13     stack_seg dd 0 ;用于接收堆栈段选择子#0x08
14     stack_len dd 1 ;程序建议的堆栈大小#0x0c
15 ;以4KB为单位
16
17     prgentry dd start ;程序入口#0x10
18     code_seg dd section.code.start ;代码段位置#0x14
19     code_len dd code_end ;代码段长度#0x18
20
21     data_seg dd section.data.start ;数据段位置#0x1c
22     data_len dd data_end ;数据段长度#0x20
23 ;-----
24 ;符号地址检索表
25 salt_items dd (header_end-salt)/256 ;#0x24
26
27 salt: ;#0x28
28 PrintString db '@PrintString'
29 times 256-($-PrintString) db 0
30
31 TerminateProgram db '@TerminateProgram'
32 times 256-($-TerminateProgram) db 0
33
34 ReadDiskData db '@ReadDiskData'
35 times 256-($-ReadDiskData) db 0
36
37 header_end:
38
39 ;=====
40 SECTION data vstart=0
41
42     message_1 db 0x0d,0x0a
43               db '[USER TASK]: Hi! nice to meet you,'
44               db 'I am run at CPL=',0
45
46     message_2 db 0
47               db '.Now,I must exit...',0x0d,0x0a,0
48
49 data_end:
50
51 ;=====
52 [bits 32]
53 ;=====
54 SECTION code vstart=0
55 start:
56 ;任务启动时, DS指向头部段, 也不需要设置堆栈
57 mov eax,ds
58 mov fs,eax
59
60 mov eax,[data_seg]
61 mov ds,eax
62
63 mov ebx,message_1
64 call far [fs:PrintString]
65
66 mov ax,cs
67 and al,0000_0011B
68 or al,0x0030
69 mov [message_2],al
70
71 mov ebx,message_2
72 call far [fs:PrintString]
73
74 call far [fs:TerminateProgram] ;退出, 并将控制权返回到核心
75
76 code_end:
77
78 ;-----
79 SECTION trail
80 ;-----
81 program_end:
```





```

1      ;代码清单15-1
2      ;文件名: c15_core.asm
3      ;文件说明: 保护模式微型核心程序
4      ;创建日期: 2011-11-19 21:40
5
6      ;以下常量定义部分。内核的大部分内容都应当固定
7      core_code_seg_sel    equ    0x38    ;内核代码段选择子
8      core_data_seg_sel    equ    0x30    ;内核数据段选择子
9      sys_routine_seg_sel  equ    0x28    ;系统公共例程代码段的选择子
10     video_ram_seg_sel    equ    0x20    ;视频显示缓冲区的段选择子
11     core_stack_seg_sel   equ    0x18    ;内核堆栈段选择子
12     mem_0_4_gb_seg_sel   equ    0x08    ;整个0-4GB内存的段的选择子
13
14     ;-----
15     ;以下是系统核心的头部, 用于加载核心程序
16     core_length          dd    core_end    ;核心程序总长度#00
17
18     sys_routine_seg      dd    section.sys_routine.start
19                             ;系统公用例程段位置#04
20
21     core_data_seg        dd    section.core_data.start
22                             ;核心数据段位置#08
23
24     core_code_seg        dd    section.core_code.start
25                             ;核心代码段位置#0c
26
27
28     core_entry            dd    start        ;核心代码段入口点#10
29                             dw    core_code_seg_sel
30
31     ;=====
32     [bits 32]
33     ;=====
34     SECTION sys_routine vstart=0            ;系统公共例程代码段
35     ;-----
36     ;字符串显示例程
37     put_string:                ;显示0终止的字符串并移动光标
38                                     ;输入: DS:EBX=串地址
39     push ecx
40     .getc:
41         mov cl,[ebx]
42         or cl,cl
43         jz .exit
44         call put_char
45         inc ebx
46         jmp .getc
47
48     .exit:
49         pop ecx
50         retf                    ;段间返回
51
52     ;-----
53     put_char:                ;在当前光标处显示一个字符, 并推进
54                                     ;光标。仅用于段内调用
55                                     ;输入: CL=字符ASCII码
56     pushad
57
58     ;以下取当前光标位置
59     mov dx,0x3d4
60     mov al,0x0e
61     out dx,al
62     inc dx                    ;0x3d5
63     in al,dx                  ;高字
64     mov ah,al
65
66     dec dx                    ;0x3d4
67     mov al,0x0f
68     out dx,al
69     inc dx                    ;0x3d5
70     in al,dx                  ;低字
71     mov bx,ax                 ;BX=代表光标位置的16位数
72
73     cmp cl,0x0d                ;回车符?
74     jnz .put_0a
75     mov ax,bx
76     mov bl,80
77     div bl
78     mul bl
79     mov bx,ax
80     jmp .set_cursor
81
82     .put_0a:
83         cmp cl,0x0a                ;换行符?
84         jnz .put_other
85         add bx,80
86         jmp .roll_screen
87
88     .put_other:
89                                     ;正常显示字符
90     push es
91     mov eax,video_ram_seg_sel    ;0xb8000段的选择子
92     mov es,eax
93     shl bx,1

```

```

93         mov [es:bx],cl
94         pop es
95
96         ;以下将光标位置推进一个字符
97         shr bx,1
98         inc bx
99
100        .roll_screen:
101        cmp bx,2000                ;光标超出屏幕? 滚屏
102        jl .set_cursor
103
104        push ds
105        push es
106        mov eax,video_ram_seg_sel
107        mov ds,eax
108        mov es,eax
109        cld
110        mov esi,0xa0                ;小心! 32位模式下movsb/w/d
111        mov edi,0x00                ;使用的是esi/edi/ecx
112        mov ecx,1920
113        rep movsd
114        mov bx,3840                ;清除屏幕最底一行
115        mov ecx,80                 ;32位程序应该使用ECX
116
117        .cls:
118        mov word[es:bx],0x0720
119        add bx,2
120        loop .cls
121
122        pop es
123        pop ds
124
125        mov bx,1920
126
127        .set_cursor:
128        mov dx,0x3d4
129        mov al,0x0e
130        out dx,al
131        inc dx                        ;0x3d5
132        mov al,bh
133        out dx,al
134        dec dx                        ;0x3d4
135        mov al,0x0f
136        out dx,al
137        inc dx                        ;0x3d5
138        mov al,bl
139        out dx,al
140
141        popad
142
143        ret
144
145        ;-----
146        read_hard_disk_0:                ;从硬盘读取一个逻辑扇区
147                                         ;EAX=逻辑扇区号
148                                         ;DS:EBX=目标缓冲区地址
149                                         ;返回: EBX=EBX+512
150
151        push eax
152        push ecx
153        push edx
154
155        push eax
156
157        mov dx,0x1f2
158        mov al,1
159        out dx,al                    ;读取的扇区数
160
161        inc dx                        ;0x1f3
162        pop eax
163        out dx,al                    ;LBA地址7~0
164
165        inc dx                        ;0x1f4
166        mov cl,8
167        shr eax,cl
168        out dx,al                    ;LBA地址15~8
169
170        inc dx                        ;0x1f5
171        shr eax,cl
172        out dx,al                    ;LBA地址23~16
173
174        inc dx                        ;0x1f6
175        shr eax,cl
176        or al,0xe0                    ;第一硬盘 LBA地址27~24
177        out dx,al
178
179        inc dx                        ;0x1f7
180        mov al,0x20                    ;读命令
181        out dx,al
182
183        .waits:
184        in al,dx
185        and al,0x88
186        cmp al,0x08

```

```

185         jnz .waits                ;不忙，且硬盘已准备好数据传输
186
187         mov ecx,256                ;总共要读取的字数
188         mov dx,0x1f0
189     .readw:
190         in ax,dx
191         mov [ebx],ax
192         add ebx,2
193         loop .readw
194
195         pop edx
196         pop ecx
197         pop eax
198
199         retf                        ;段间返回
200
201 ;-----
202 ;汇编语言程序是极难一次成功，而且调试非常困难。这个例程可以提供帮助
203 put_hex_dword:                    ;在当前光标处以十六进制形式显示
204                                   ;一个双字并推进光标
205                                   ;输入：EDX=要转换并显示的数字
206                                   ;输出：无
207
208         pushad
209         push ds
210
211         mov ax,core_data_seg_sel    ;切换到核心数据段
212         mov ds,ax
213
214         mov ebx,bin_hex              ;指向核心数据段内的转换表
215         mov ecx,8
216     .xlt:
217         rol edx,4
218         mov eax,edx
219         and eax,0x0000000f
220         xlat
221
222         push ecx
223         mov cl,al
224         call put_char
225         pop ecx
226
227         loop .xlt
228
229         pop ds
230         popad
231         retf
232 ;-----
233 allocate_memory:                  ;分配内存
234                                   ;输入：ECX=希望分配的字节数
235                                   ;输出：ECX=起始线性地址
236
237         push ds
238         push eax
239         push ebx
240
241         mov eax,core_data_seg_sel
242         mov ds,eax
243
244         mov eax,[ram_alloc]
245         add eax,ecx                  ;下一次分配时的起始地址
246
247         ;这里应当有检测可用内存数量的指令
248
249         mov ecx,[ram_alloc]          ;返回分配的起始地址
250
251         mov ebx,eax
252         and ebx,0xffffffffc
253         add ebx,4                    ;强制对齐
254         test eax,0x00000003          ;下次分配的起始地址最好是4字节对齐
255         cmovnz eax,ebx               ;如果没有对齐，则强制对齐
256         mov [ram_alloc],eax          ;下次从该地址分配内存
257                                   ;cmovcc指令可以避免控制转移
258         pop ebx
259         pop eax
260         pop ds
261
262         retf
263 ;-----
264 set_up_gdt_descriptor:            ;在GDT内安装一个新的描述符
265                                   ;输入：EDX:EAX=描述符
266                                   ;输出：CX=描述符的选择子
267
268         push eax
269         push ebx
270         push edx
271
272         push ds
273         push es
274
275         mov ebx,core_data_seg_sel    ;切换到核心数据段
276         mov ds,ebx
277

```

```

277         sgdt [pgdt]                ;以便开始处理GDT
278
279         mov ebx,mem_0_4_gb_seg_sel
280         mov es,ebx
281
282         movzx ebx,word [pgdt]        ;GDT界限
283         inc bx                       ;GDT总字节数，也是下一个描述符偏移
284         add ebx,[pgdt+2]             ;下一个描述符的线性地址
285
286         mov [es:ebx],eax
287         mov [es:ebx+4],edx
288
289         add word [pgdt],8            ;增加一个描述符的大小
290
291         lgdt [pgdt]                 ;对GDT的更改生效
292
293         mov ax,[pgdt]               ;得到GDT界限值
294         xor dx,dx
295         mov bx,8
296         div bx                       ;除以8，去掉余数
297         mov cx,ax
298         shl cx,3                    ;将索引号移到正确位置
299
300         pop es
301         pop ds
302
303         pop edx
304         pop ebx
305         pop eax
306
307         retf
308 ;-----
309 make_seg_descriptor:                ;构造存储器和系统的段描述符
310                                     ;输入: EAX=线性基地址
311                                     ;      EBX=段界限
312                                     ;      ECX=属性。各属性位都在原始
313                                     ;      位置，无关的位清零
314                                     ;返回: EDX:EAX=描述符
315         mov edx,eax
316         shl eax,16
317         or ax,bx                    ;描述符前32位(EAX)构造完毕
318
319         and edx,0xffff0000          ;清除基地址中无关的位
320         rol edx,8
321         bswap edx                   ;装配基址的31~24和23~16 (80486+)
322
323         xor bx,bx
324         or edx,ebx                  ;装配段界限的高4位
325
326         or edx,ecx                  ;装配属性
327
328         retf
329 ;-----
330 make_gate_descriptor:               ;构造门的描述符(调用门等)
331                                     ;输入: EAX=门代码在段内偏移地址
332                                     ;      BX=门代码所在段的选择子
333                                     ;      CX=段类型及属性等(各属
334                                     ;      性位都在原始位置)
335                                     ;返回: EDX:EAX=完整的描述符
336
337         push ebx
338         push ecx
339
340         mov edx,eax
341         and edx,0xffff0000          ;得到偏移地址高16位
342         or dx,cx                    ;组装属性部分到EDX
343
344         and eax,0x0000ffff          ;得到偏移地址低16位
345         shl ebx,16
346         or eax,ebx                  ;组装段选择子部分
347
348         pop ecx
349         pop ebx
350
351         retf
352 ;-----
353 terminate_current_task:             ;终止当前任务
354                                     ;注意，执行此例程时，当前任务仍在
355                                     ;运行中。此例程其实也是当前任务的
356                                     ;一部分
357
358         pushfd
359         mov edx,[esp]               ;获得EFLAGS寄存器内容
360         add esp,4                   ;恢复堆栈指针
361
362         mov eax,core_data_seg_sel
363         mov ds,eax
364
365         test dx,0100_0000_0000_0000B ;测试NT位
366         jnz .b1                     ;当前任务是嵌套的，到.b1执行iretd
367         mov ebx,core_msg1           ;当前任务不是嵌套的，直接切换到
368         call sys_routine_seg_sel:put_string

```

```

369         jmp far [prgman_tss]                ;程序管理器任务
370
371     .b1:
372         mov ebx,core_msg0
373         call sys_routine_seg_sel:put_string
374         iretd
375
376 sys_routine_end:
377
378 ;=====
379 SECTION core_data vstart=0                ;系统核心的数据段
380 ;-----
381         pgdt            dw 0                ;用于设置和修改GDT
382                        dd 0
383
384         ram_alloc       dd 0x00100000      ;下次分配内存时的起始地址
385
386 ;符号地址检索表
387 salt:
388 salt_1                db '@PrintString'
389                        times 256-($-salt_1) db 0
390                        dd put_string
391                        dw sys_routine_seg_sel
392
393 salt_2                db '@ReadDiskData'
394                        times 256-($-salt_2) db 0
395                        dd read_hard_disk_0
396                        dw sys_routine_seg_sel
397
398 salt_3                db '@PrintDwordAsHexString'
399                        times 256-($-salt_3) db 0
400                        dd put_hex_dword
401                        dw sys_routine_seg_sel
402
403 salt_4                db '@TerminateProgram'
404                        times 256-($-salt_4) db 0
405                        dd terminate_current_task
406                        dw sys_routine_seg_sel
407
408 salt_item_len         equ $-salt_4
409 salt_items            equ ($-salt)/salt_item_len
410
411 message_1             db ' If you seen this message,that means we '
412                        db 'are now in protect mode,and the system '
413                        db 'core is loaded,and the video display '
414                        db 'routine works perfectly.',0x0d,0x0a,0
415
416 message_2             db ' System wide CALL-GATE mounted.',0x0d,0x0a,0
417
418 bin_hex               db '0123456789ABCDEF'
419                        ;put_hex_dword子过程用的查找表
420
421 core_buf              times 2048 db 0      ;内核用的缓冲区
422
423 cpu_brnd0             db 0x0d,0x0a,' ',0
424 cpu_brand             times 52 db 0
425 cpu_brnd1             db 0x0d,0x0a,0x0d,0x0a,0
426
427 ;任务控制块链
428 tcb_chain             dd 0
429
430 ;程序管理器的任务信息
431 prgman_tss            dd 0                ;程序管理器的TSS基地址
432                        dw 0                ;程序管理器的TSS描述符选择子
433
434 prgman_msg1           db 0x0d,0x0a
435                        db '[PROGRAM MANAGER]: Hello! I am Program Manager,'
436                        db 'run at CPL=0.Now,create user task and switch '
437                        db 'to it by the CALL instruction...',0x0d,0x0a,0
438
439 prgman_msg2           db 0x0d,0x0a
440                        db '[PROGRAM MANAGER]: I am glad to regain control.'
441                        db 'Now,create another user task and switch to '
442                        db 'it by the JMP instruction...',0x0d,0x0a,0
443
444 prgman_msg3           db 0x0d,0x0a
445                        db '[PROGRAM MANAGER]: I am gain control again,'
446                        db 'HALT...',0
447
448 core_msg0             db 0x0d,0x0a
449                        db '[SYSTEM CORE]: Uh...This task initiated with '
450                        db 'CALL instruction or an exeception/ interrupt,'
451                        db 'should use IRETD instruction to switch back...'
452                        db 0x0d,0x0a,0
453
454 core_msg1             db 0x0d,0x0a
455                        db '[SYSTEM CORE]: Uh...This task initiated with '
456                        db 'JMP instruction, should switch to Program '
457                        db 'Manager directly by the JMP instruction...'
458                        db 0x0d,0x0a,0
459
460 core_data_end:

```

```

461
462 ;=====
463 SECTION core_code vstart=0
464 ;-----
465 fill_descriptor_in_ldt:                ;在LDT内安装一个新的描述符
466                                     ;输入: EDX:EAX=描述符
467                                     ;      EBX=TCB基地址
468                                     ;输出: CX=描述符的选择子
469
470     push eax
471     push edx
472     push edi
473     push ds
474
475     mov ecx,mem_0_4_gb_seg_sel
476     mov ds,ecx
477
478     mov edi,[ebx+0x0c]                ;获得LDT基地址
479
480     xor ecx,ecx
481     mov cx,[ebx+0x0a]                ;获得LDT界限
482     inc cx                          ;LDT的总字节数, 即新描述符偏移地址
483
484     mov [edi+ecx+0x00],eax
485     mov [edi+ecx+0x04],edx          ;安装描述符
486
487     add cx,8
488     dec cx                          ;得到新的LDT界限值
489
490     mov [ebx+0x0a],cx              ;更新LDT界限值到TCB
491
492     mov ax,cx
493     xor dx,dx
494     mov cx,8
495     div cx
496
497     mov cx,ax
498     shl cx,3                        ;左移3位, 并且
499     or cx,0000_0000_0000_0100B    ;使TI位=1, 指向LDT, 最后使RPL=00
500
501     pop ds
502     pop edi
503     pop edx
504     pop eax
505
506     ret
507 ;-----
508 load_relocate_program:                ;加载并重定位用户程序
509                                     ;输入: PUSH 逻辑扇区号
510                                     ;      PUSH 任务控制块基地址
511                                     ;输出: 无
512
513     pushad
514
515     push ds
516     push es
517
518     mov ebp,esp                    ;为访问通过堆栈传递的参数做准备
519
520     mov ecx,mem_0_4_gb_seg_sel
521     mov es,ecx
522
523     mov esi,[ebp+11*4]            ;从堆栈中取得TCB的基地址
524
525     ;以下申请创建LDT所需要的内存
526     mov ecx,160                    ;允许安装20个LDT描述符
527     call sys_routine_seg_sel:allocate_memory
528     mov [es:esi+0x0c],ecx          ;登记LDT基地址到TCB中
529     mov word [es:esi+0x0a],0xffff  ;登记LDT初始的界限到TCB中
530
531     ;以下开始加载用户程序
532     mov eax,core_data_seg_sel
533     mov ds,eax                    ;切换DS到内核数据段
534
535     mov eax,[ebp+12*4]            ;从堆栈中取出用户程序起始扇区号
536     mov ebx,core_buf
537     call sys_routine_seg_sel:read_hard_disk_0
538
539     ;以下判断整个程序有多大
540     mov eax,[core_buf]            ;程序尺寸
541     mov ebx,eax
542     and ebx,0xffffffe0            ;使之512字节对齐 (能被512整除的数低
543     add ebx,512                    ;9位都为0
544     test eax,0x000001ff           ;程序的大小正好是512的倍数吗?
545     cmovnz eax,ebx                ;不是。使用凑整的结果
546
547     mov ecx,eax                    ;实际需要申请的内存数量
548     call sys_routine_seg_sel:allocate_memory
549     mov [es:esi+0x06],ecx          ;登记程序加载基地址到TCB中
550
551     mov ebx,ecx                    ;ebx -> 申请到的内存首地址
552     xor edx,edx
553     mov ecx,512

```

```

553         div ecx
554         mov ecx,eax                ;总扇区数
555
556         mov eax,mem_0_4_gb_seg_sel ;切换DS到0-4GB的段
557         mov ds,eax
558
559         mov eax,[ebp+12*4]          ;起始扇区号
560     .b1:
561         call sys_routine_seg_sel:read_hard_disk_0
562         inc eax
563         loop .b1                    ;循环读，直到读完整个用户程序
564
565         mov edi,[es:esi+0x06]        ;获得程序加载基地址
566
567         ;建立程序头部段描述符
568         mov eax,edi                  ;程序头部起始线性地址
569         mov ebx,[edi+0x04]           ;段长度
570         dec ebx                      ;段界限
571         mov ecx,0x0040f200           ;字节粒度的数据段描述符，特权级3
572         call sys_routine_seg_sel:make_seg_descriptor
573
574         ;安装头部段描述符到LDT中
575         mov ebx,esi                  ;TCB的基地址
576         call fill_descriptor_in_ldt
577
578         or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
579         mov [es:esi+0x44],cx         ;登记程序头部段选择子到TCB
580         mov [edi+0x04],cx            ;和头部内
581
582         ;建立程序代码段描述符
583         mov eax,edi
584         add eax,[edi+0x14]            ;代码起始线性地址
585         mov ebx,[edi+0x18]            ;段长度
586         dec ebx                      ;段界限
587         mov ecx,0x0040f800           ;字节粒度的代码段描述符，特权级3
588         call sys_routine_seg_sel:make_seg_descriptor
589         mov ebx,esi                  ;TCB的基地址
590         call fill_descriptor_in_ldt
591         or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
592         mov [edi+0x14],cx            ;登记代码段选择子到头部
593
594         ;建立程序数据段描述符
595         mov eax,edi
596         add eax,[edi+0x1c]            ;数据段起始线性地址
597         mov ebx,[edi+0x20]            ;段长度
598         dec ebx                      ;段界限
599         mov ecx,0x0040f200           ;字节粒度的数据段描述符，特权级3
600         call sys_routine_seg_sel:make_seg_descriptor
601         mov ebx,esi                  ;TCB的基地址
602         call fill_descriptor_in_ldt
603         or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
604         mov [edi+0x1c],cx            ;登记数据段选择子到头部
605
606         ;建立程序堆栈段描述符
607         mov ecx,[edi+0x0c]            ;4KB的倍率
608         mov ebx,0x000fffff
609         sub ebx,ecx                  ;得到段界限
610         mov eax,4096
611         mul ecx
612         mov ecx,eax                  ;准备为堆栈分配内存
613         call sys_routine_seg_sel:allocate_memory
614         add eax,ecx                  ;得到堆栈的高端物理地址
615         mov ecx,0x00c0f600           ;字节粒度的堆栈段描述符，特权级3
616         call sys_routine_seg_sel:make_seg_descriptor
617         mov ebx,esi                  ;TCB的基地址
618         call fill_descriptor_in_ldt
619         or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
620         mov [edi+0x08],cx            ;登记堆栈段选择子到头部
621
622         ;重定位SALT
623         mov eax,mem_0_4_gb_seg_sel ;这里和前一章不同，头部段描述符
624         mov es,eax                  ;已安装，但还没有生效，故只能通
625                                     ;过4GB段访问用户程序头部
626         mov eax,core_data_seg_sel
627         mov ds,eax
628
629         cld
630
631         mov ecx,[es:edi+0x24]         ;U-SALT条目数(通过访问4GB段取得)
632         add edi,0x28                 ;U-SALT在4GB段内的偏移
633     .b2:
634         push ecx
635         push edi
636
637         mov ecx,salt_items
638         mov esi,salt
639     .b3:
640         push edi
641         push esi
642         push ecx
643
644         mov ecx,64                    ;检索表中，每条目的比较次数

```

```

645         repe cmpsd                ;每次比较4字节
646         jnz .b4
647         mov eax,[esi]              ;若匹配,则esi恰好指向其后的地址
648         mov [es:edi-256],eax       ;将字符串改写成偏移地址
649         mov ax,[esi+4]
650         or ax,0000000000000011B   ;以用户程序自己的特权级使用调用门
651                                     ;故RPL=3
652         mov [es:edi-252],ax        ;回填调用门选择子
653     .b4:
654
655         pop ecx
656         pop esi
657         add esi,salt_item_len
658         pop edi                    ;从头比较
659         loop .b3
660
661         pop edi
662         add edi,256
663         pop ecx
664         loop .b2
665
666         mov esi,[ebp+11*4]          ;从堆栈中取得TCB的基地址
667
668         ;创建0特权级堆栈
669         mov ecx,4096
670         mov eax,ecx                ;为生成堆栈高端地址做准备
671         mov [es:esi+0x1a],ecx
672         shr dword [es:esi+0x1a],12 ;登记0特权级堆栈尺寸到TCB
673         call sys_routine_seg_sel:allocate_memory
674         add eax,ecx                ;堆栈必须使用高端地址为基地址
675         mov [es:esi+0x1e],eax      ;登记0特权级堆栈基地址到TCB
676         mov ebx,0xfffffe           ;段长度(界限)
677         mov ecx,0x00c09600         ;4KB粒度,读写,特权级0
678         call sys_routine_seg_sel:make_seg_descriptor
679         mov ebx,esi                ;TCB的基地址
680         call fill_descriptor_in_ldt
681         ;or cx,0000_0000_0000_0000 ;设置选择子的特权级为0
682         mov [es:esi+0x22],cx       ;登记0特权级堆栈选择子到TCB
683         mov dword [es:esi+0x24],0  ;登记0特权级堆栈初始ESP到TCB
684
685         ;创建1特权级堆栈
686         mov ecx,4096
687         mov eax,ecx                ;为生成堆栈高端地址做准备
688         mov [es:esi+0x28],ecx
689         shr [es:esi+0x28],12       ;登记1特权级堆栈尺寸到TCB
690         call sys_routine_seg_sel:allocate_memory
691         add eax,ecx                ;堆栈必须使用高端地址为基地址
692         mov [es:esi+0x2c],eax      ;登记1特权级堆栈基地址到TCB
693         mov ebx,0xfffffe           ;段长度(界限)
694         mov ecx,0x00c0b600         ;4KB粒度,读写,特权级1
695         call sys_routine_seg_sel:make_seg_descriptor
696         mov ebx,esi                ;TCB的基地址
697         call fill_descriptor_in_ldt
698         ;or cx,0000_0000_0000_0001 ;设置选择子的特权级为1
699         mov [es:esi+0x30],cx       ;登记1特权级堆栈选择子到TCB
700         mov dword [es:esi+0x32],0  ;登记1特权级堆栈初始ESP到TCB
701
702         ;创建2特权级堆栈
703         mov ecx,4096
704         mov eax,ecx                ;为生成堆栈高端地址做准备
705         mov [es:esi+0x36],ecx
706         shr [es:esi+0x36],12       ;登记2特权级堆栈尺寸到TCB
707         call sys_routine_seg_sel:allocate_memory
708         add eax,ecx                ;堆栈必须使用高端地址为基地址
709         mov [es:esi+0x3a],ecx      ;登记2特权级堆栈基地址到TCB
710         mov ebx,0xfffffe           ;段长度(界限)
711         mov ecx,0x00c0d600         ;4KB粒度,读写,特权级2
712         call sys_routine_seg_sel:make_seg_descriptor
713         mov ebx,esi                ;TCB的基地址
714         call fill_descriptor_in_ldt
715         ;or cx,0000_0000_0000_0010 ;设置选择子的特权级为2
716         mov [es:esi+0x3e],cx       ;登记2特权级堆栈选择子到TCB
717         mov dword [es:esi+0x40],0  ;登记2特权级堆栈初始ESP到TCB
718
719         ;在GDT中登记LDT描述符
720         mov eax,[es:esi+0x0c]       ;LDT的起始线性地址
721         movzx ebx,word [es:esi+0x0a] ;LDT段界限
722         mov ecx,0x00408200          ;LDT描述符,特权级0
723         call sys_routine_seg_sel:make_seg_descriptor
724         call sys_routine_seg_sel:set_up_gdt_descriptor
725         mov [es:esi+0x10],cx        ;登记LDT选择子到TCB中
726
727         ;创建用户程序的TSS
728         mov ecx,104                 ;tss的基本尺寸
729         mov [es:esi+0x12],cx
730         dec word [es:esi+0x12]      ;登记TSS界限值到TCB
731         call sys_routine_seg_sel:allocate_memory
732         mov [es:esi+0x14],ecx       ;登记TSS基地址到TCB
733
734         ;登记基本的TSS表格内容
735         mov word [es:ecx+0],0       ;反向链=0
736

```



```

737      mov edx,[es:esi+0x24]          ;登记0特权级堆栈初始ESP
738      mov [es:ecx+4],edx            ;到TSS中
739
740      mov dx,[es:esi+0x22]           ;登记0特权级堆栈段选择子
741      mov [es:ecx+8],dx             ;到TSS中
742
743      mov edx,[es:esi+0x32]           ;登记1特权级堆栈初始ESP
744      mov [es:ecx+12],edx           ;到TSS中
745
746      mov dx,[es:esi+0x30]           ;登记1特权级堆栈段选择子
747      mov [es:ecx+16],dx           ;到TSS中
748
749      mov edx,[es:esi+0x40]           ;登记2特权级堆栈初始ESP
750      mov [es:ecx+20],edx           ;到TSS中
751
752      mov dx,[es:esi+0x3e]           ;登记2特权级堆栈段选择子
753      mov [es:ecx+24],dx           ;到TSS中
754
755      mov dx,[es:esi+0x10]           ;登记任务的LDT选择子
756      mov [es:ecx+96],dx           ;到TSS中
757
758      mov dx,[es:esi+0x12]           ;登记任务的I/O位图偏移
759      mov [es:ecx+102],dx          ;到TSS中
760
761      mov word [es:ecx+100],0         ;T=0
762
763      mov dword [es:ecx+28],0        ;登记CR3(PDBR)
764
765      ;访问用户程序头部，获取数据填充TSS
766      mov ebx,[ebp+11*4]             ;从堆栈中取得TCB的基地址
767      mov edi,[es:ebx+0x06]          ;用户程序加载的基地址
768
769      mov edx,[es:edi+0x10]           ;登记程序入口点（EIP）
770      mov [es:ecx+32],edx           ;到TSS
771
772      mov dx,[es:edi+0x14]           ;登记程序代码段（CS）选择子
773      mov [es:ecx+76],dx           ;到TSS中
774
775      mov dx,[es:edi+0x08]           ;登记程序堆栈段（SS）选择子
776      mov [es:ecx+80],dx           ;到TSS中
777
778      mov dx,[es:edi+0x04]           ;登记程序数据段（DS）选择子
779      mov word [es:ecx+84],dx        ;到TSS中。注意，它指向程序头部段
780
781      mov word [es:ecx+72],0         ;TSS中的ES=0
782
783      mov word [es:ecx+88],0         ;TSS中的FS=0
784
785      mov word [es:ecx+92],0         ;TSS中的GS=0
786
787      pushfd
788      pop edx
789
790      mov dword [es:ecx+36],edx       ;EFLAGS
791
792      ;在GDT中登记TSS描述符
793      mov eax,[es:esi+0x14]           ;TSS的起始线性地址
794      movzx ebx,word [es:esi+0x12]    ;段长度（界限）
795      mov ecx,0x00408900             ;TSS描述符，特权级0
796      call sys_routine_seg_sel:make_seg_descriptor
797      call sys_routine_seg_sel:set_up_gdt_descriptor
798      mov [es:esi+0x18],cx           ;登记TSS选择子到TCB
799
800      pop es                         ;恢复到调用此过程前的es段
801      pop ds                         ;恢复到调用此过程前的ds段
802
803      popad
804
805      ret 8                          ;丢弃调用本过程前压入的参数
806
807      ;-----
808      append_to_tcb_link:            ;在TCB链上追加任务控制块
809                                     ;输入：ECX=TCB线性基地址
810      push eax
811      push edx
812      push ds
813      push es
814
815      mov eax,core_data_seg_sel       ;令DS指向内核数据段
816      mov ds,edx
817      mov eax,mem_0_4_gb_seg_sel     ;令ES指向0..4GB段
818      mov es,edx
819
820      mov dword [es: ecx+0x00],0      ;当前TCB指针域清零，以指示这是最
821                                     ;后一个TCB
822
823      mov eax,[tcb_chain]             ;TCB表头指针
824      or eax,ecx                     ;链表为空？
825      jz .notcb
826
827      .searc:
828      mov edx,eax

```

```

829         mov eax,[es: edx+0x00]
830         or  eax,eax
831         jnz .searc
832
833         mov [es: edx+0x00],ecx
834         jmp .retpc
835
836 .notcb:
837         mov [tcb_chain],ecx          ;若为空表，直接令表头指针指向TCB
838
839 .retpc:
840         pop es
841         pop ds
842         pop edx
843         pop eax
844
845         ret
846
847 ;-----
848 start:
849         mov ecx,core_data_seg_sel    ;令DS指向核心数据段
850         mov ds,ecx
851
852         mov ecx,mem_0_4_gb_seg_sel   ;令ES指向4GB数据段
853         mov es,ecx
854
855         mov ebx,message_1
856         call sys_routine_seg_sel:put_string
857
858         ;显示处理器品牌信息
859         mov eax,0x80000002
860         cpuid
861         mov [cpu_brand + 0x00],eax
862         mov [cpu_brand + 0x04],ebx
863         mov [cpu_brand + 0x08],ecx
864         mov [cpu_brand + 0x0c],edx
865
866         mov eax,0x80000003
867         cpuid
868         mov [cpu_brand + 0x10],eax
869         mov [cpu_brand + 0x14],ebx
870         mov [cpu_brand + 0x18],ecx
871         mov [cpu_brand + 0x1c],edx
872
873         mov eax,0x80000004
874         cpuid
875         mov [cpu_brand + 0x20],eax
876         mov [cpu_brand + 0x24],ebx
877         mov [cpu_brand + 0x28],ecx
878         mov [cpu_brand + 0x2c],edx
879
880         mov ebx,cpu_brnd0            ;显示处理器品牌信息
881         call sys_routine_seg_sel:put_string
882         mov ebx,cpu_brand
883         call sys_routine_seg_sel:put_string
884         mov ebx,cpu_brnd1
885         call sys_routine_seg_sel:put_string
886
887         ;以下开始安装为整个系统服务的调用门。特权级之间的控制转移必须使用门
888         mov edi,salt                 ;C-SALT表的起始位置
889         mov ecx,salt_items           ;C-SALT表的条目数量
890
891 .b3:
892         push ecx
893         mov eax,[edi+256]             ;该条目入口点的32位偏移地址
894         mov bx,[edi+260]             ;该条目入口点的段选择子
895         mov cx,1_11_0_1100_000_00000B ;特权级3的调用门(3以上的特权级才
896                                     ;允许访问)，0个参数(因为用寄存器
897                                     ;传递参数，而没有用栈)
898         call sys_routine_seg_sel:make_gate_descriptor
899         call sys_routine_seg_sel:set_up_gdt_descriptor
900         mov [edi+260],cx             ;将返回的门描述符选择子回填
901         add edi,salt_item_len        ;指向下一个C-SALT条目
902         pop ecx
903         loop .b3
904
905         ;对门进行测试
906         mov ebx,message_2
907         call far [salt_1+256]         ;通过门显示信息(偏移量将被忽略)
908
909         ;为程序管理器的TSS分配内存空间
910         mov ecx,104                  ;为该任务的TSS分配内存
911         call sys_routine_seg_sel:allocate_memory
912         mov [prgman_tss+0x00],ecx    ;保存程序管理器的TSS基地址
913
914         ;在程序管理器的TSS中设置必要的项目
915         mov word [es:ecx+96],0        ;没有LDT。处理器允许没有LDT的任务。
916         mov word [es:ecx+102],103     ;没有I/O位图。0特权级事实上不需要。
917         mov word [es:ecx+0],0         ;反向链=0
918         mov dword [es:ecx+28],0       ;登记CR3(PDBR)
919         mov word [es:ecx+100],0       ;T=0
920                                     ;不需要0、1、2特权级堆栈。0特权不
921                                     ;会向低特权级转移控制。

```

```

921
922 ;创建TSS描述符，并安装到GDT中
923 mov eax,ecx ;TSS的起始线性地址
924 mov ebx,103 ;段长度（界限）
925 mov ecx,0x00408900 ;TSS描述符，特权级0
926 call sys_routine_seg_sel:make_seg_descriptor
927 call sys_routine_seg_sel:set_up_gdt_descriptor
928 mov [prgman_tss+0x04],cx ;保存程序管理器的TSS描述符选择子
929
930 ;任务寄存器TR中的内容是任务存在的标志，该内容也决定了当前任务是谁。
931 ;下面的指令为当前正在执行的0特权级任务“程序管理器”后补手续（TSS）。
932 ltr cx
933
934 ;现在可认为“程序管理器”任务正执行中
935 mov ebx,prgman_msg1
936 call sys_routine_seg_sel:put_string
937
938 mov ecx,0x46
939 call sys_routine_seg_sel:allocate_memory
940 call append_to_tcb_link ;将此TCB添加到TCB链中
941
942 push dword 50 ;用户程序位于逻辑50扇区
943 push ecx ;压入任务控制块起始线性地址
944
945 call load_relocate_program
946
947 call far [es:ecx+0x14] ;执行任务切换。和上一章不同，任务切
948 ;换时要恢复TSS内容，所以在创建任务
949 ;时TSS要填写完整
950
951 ;重新加载并切换任务
952 mov ebx,prgman_msg2
953 call sys_routine_seg_sel:put_string
954
955 mov ecx,0x46
956 call sys_routine_seg_sel:allocate_memory
957 call append_to_tcb_link ;将此TCB添加到TCB链中
958
959 push dword 50 ;用户程序位于逻辑50扇区
960 push ecx ;压入任务控制块起始线性地址
961
962 call load_relocate_program
963
964 jmp far [es:ecx+0x14] ;执行任务切换
965
966 mov ebx,prgman_msg3
967 call sys_routine_seg_sel:put_string
968
969 hlt
970
971 core_code_end:
972
973 ;-----
974 SECTION core_trail
975 ;-----
976 core_end:

```



```
1      ;代码清单16-2
2      ;文件名: c16.asm
3      ;文件说明: 用户程序
4      ;创建日期: 2012-05-25 13:53
5
6      program_length dd program_end      ;程序总长度#0x00
7      entry_point    dd start           ;程序入口点#0x04
8      salt_position  dd salt_begin      ;SALT表起始偏移量#0x08
9      salt_items     dd (salt_end-salt_begin)/256 ;SALT条目数#0x0C
10
11 ;-----
12
13 ;符号地址检索表
14 salt_begin:
15
16     PrintString     db '@PrintString'
17                     times 256-($-PrintString) db 0
18
19     TerminateProgram db '@TerminateProgram'
20                     times 256-($-TerminateProgram) db 0
21 ;-----
22
23     reserved times 256*500 db 0          ;保留一个空白区, 以演示分页
24
25 ;-----
26     ReadDiskData    db '@ReadDiskData'
27                     times 256-($-ReadDiskData) db 0
28
29     PrintDwordAsHex db '@PrintDwordAsHexString'
30                     times 256-($-PrintDwordAsHex) db 0
31
32     salt_end:
33
34     message_0        db 0x0d,0x0a,
35                       db ' .....User task is running with '
36                       db 'paging enabled!.....',0x0d,0x0a,0
37
38     space            db 0x20,0x20,0
39
40 ;-----
41 [bits 32]
42 ;-----
43
44 start:
45
46     mov ebx,message_0
47     call far [PrintString]
48
49     xor esi,esi
50     mov ecx,88
51 .b1:
52     mov ebx,space
53     call far [PrintString]
54
55     mov edx,[esi*4]
56     call far [PrintDwordAsHex]
57
58     inc esi
59     loop .b1
60
61     call far [TerminateProgram]          ;退出, 并将控制权返回到核心
62
63 ;-----
64 program_end:
```



```

1      ;代码清单16-1
2      ;文件名: c16_core.asm
3      ;文件说明: 保护模式微型核心程序
4      ;创建日期: 2012-06-20 00:05
5
6      ;以下常量定义部分。内核的大部分内容都应当固定
7      core_code_seg_sel    equ 0x38    ;内核代码段选择子
8      core_data_seg_sel    equ 0x30    ;内核数据段选择子
9      sys_routine_seg_sel  equ 0x28    ;系统公共例程代码段的选择子
10     video_ram_seg_sel    equ 0x20    ;视频显示缓冲区的段选择子
11     core_stack_seg_sel   equ 0x18    ;内核堆栈段选择子
12     mem_0_4_gb_seg_sel   equ 0x08    ;整个0-4GB内存的段的选择子
13
14     ;-----
15     ;以下是系统核心的头部, 用于加载核心程序
16     core_length          dd core_end    ;核心程序总长度#00
17
18     sys_routine_seg      dd section.sys_routine.start
19                          ;系统公用例程段位置#04
20
21     core_data_seg        dd section.core_data.start
22                          ;核心数据段位置#08
23
24     core_code_seg        dd section.core_code.start
25                          ;核心代码段位置#0c
26
27
28     core_entry           dd start        ;核心代码段入口点#10
29                          dw core_code_seg_sel
30
31     ;=====
32     [bits 32]
33     ;=====
34     SECTION sys_routine vstart=0        ;系统公共例程代码段
35     ;-----
36     ;字符串显示例程
37     put_string:                      ;显示0终止的字符串并移动光标
38                                     ;输入: DS:EBX=串地址
39
40     push ecx
41
42     .getc:
43     mov cl,[ebx]
44     or cl,cl
45     jz .exit
46     call put_char
47     inc ebx
48     jmp .getc
49
50     .exit:
51     pop ecx
52     retf                            ;段间返回
53
54     ;-----
55     put_char:                      ;在当前光标处显示一个字符,并推进
56                                     ;光标。仅用于段内调用
57                                     ;输入: CL=字符ASCII码
58
59     pushad
60
61     ;以下取当前光标位置
62     mov dx,0x3d4
63     mov al,0x0e
64     out dx,al
65     inc dx                            ;0x3d5
66     in al,dx                          ;高字
67     mov ah,al
68
69     dec dx                            ;0x3d4
70     mov al,0x0f
71     out dx,al
72     inc dx                            ;0x3d5
73     in al,dx                          ;低字
74     mov bx,ax                         ;BX=代表光标位置的16位数
75
76     cmp cl,0x0d                       ;回车符?
77     jnz .put_0a
78     mov ax,bx
79     mov bl,80
80     div bl
81     mul bl
82     mov bx,ax
83     jmp .set_cursor
84
85     .put_0a:
86     cmp cl,0x0a                       ;换行符?
87     jnz .put_other
88     add bx,80
89     jmp .roll_screen
90
91     .put_other:
92     ;正常显示字符
93     push es
94     mov eax,video_ram_seg_sel        ;0x800b8000段的选择子
95     mov es,eax
96     shl bx,1

```

```

93      mov [es:bx],cl
94      pop es
95
96      ;以下将光标位置推进一个字符
97      shr bx,1
98      inc bx
99
100     .roll_screen:
101         cmp bx,2000                ;光标超出屏幕? 滚屏
102         jl .set_cursor
103
104         push ds
105         push es
106         mov eax,video_ram_seg_sel
107         mov ds,eax
108         mov es,eax
109         cld
110         mov esi,0xa0                ;小心! 32位模式下movsb/w/d
111         mov edi,0x00                ;使用的是esi/edi/ecx
112         mov ecx,1920
113         rep movsd
114         mov bx,3840                ;清除屏幕最底一行
115         mov ecx,80                  ;32位程序应该使用ECX
116     .cls:
117         mov word[es:bx],0x0720
118         add bx,2
119         loop .cls
120
121         pop es
122         pop ds
123
124         mov bx,1920
125
126     .set_cursor:
127         mov dx,0x3d4
128         mov al,0x0e
129         out dx,al
130         inc dx                        ;0x3d5
131         mov al,bh
132         out dx,al
133         dec dx                        ;0x3d4
134         mov al,0x0f
135         out dx,al
136         inc dx                        ;0x3d5
137         mov al,bl
138         out dx,al
139
140         popad
141
142         ret
143
144 ;-----
145 read_hard_disk_0:                ;从硬盘读取一个逻辑扇区
146                                ;EAX=逻辑扇区号
147                                ;DS:EBX=目标缓冲区地址
148                                ;返回: EBX=EBX+512
149
150         push eax
151         push ecx
152         push edx
153
154         push eax
155
156         mov dx,0x1f2
157         mov al,1                    ;读取的扇区数
158         out dx,al
159
160         inc dx                        ;0x1f3
161         pop eax
162         out dx,al                    ;LBA地址7~0
163
164         inc dx                        ;0x1f4
165         mov cl,8
166         shr eax,cl                    ;LBA地址15~8
167         out dx,al
168
169         inc dx                        ;0x1f5
170         shr eax,cl                    ;LBA地址23~16
171         out dx,al
172
173         inc dx                        ;0x1f6
174         shr eax,cl                    ;第一硬盘 LBA地址27~24
175         or al,0xe0
176         out dx,al
177
178         inc dx                        ;0x1f7
179         mov al,0x20                    ;读命令
180         out dx,al
181
182     .waits:
183         in al,dx
184         and al,0x88
185         cmp al,0x08

```



```

185         jnz .waits                ;不忙，且硬盘已准备好数据传输
186
187         mov ecx,256                ;总共要读取的字数
188         mov dx,0x1f0
189     .readw:
190         in ax,dx
191         mov [ebx],ax
192         add ebx,2
193         loop .readw
194
195         pop edx
196         pop ecx
197         pop eax
198
199         retf                        ;段间返回
200
201 ;-----
202 ;汇编语言程序是极难一次成功，而且调试非常困难。这个例程可以提供帮助
203 put_hex_dword:                    ;在当前光标处以十六进制形式显示
204                                   ;一个双字并推进光标
205                                   ;输入：EDX=要转换并显示的数字
206                                   ;输出：无
207         pushad
208         push ds
209
210         mov ax,core_data_seg_sel    ;切换到核心数据段
211         mov ds,ax
212
213         mov ebx,bin_hex             ;指向核心数据段内的转换表
214         mov ecx,8
215     .xlt:
216         rol edx,4
217         mov eax,edx
218         and eax,0x0000000f
219         xlat
220
221         push ecx
222         mov cl,al
223         call put_char
224         pop ecx
225
226         loop .xlt
227
228         pop ds
229         popad
230
231         retf
232
233 ;-----
234 set_up_gdt_descriptor:            ;在GDT内安装一个新的描述符
235                                   ;输入：EDX:EAX=描述符
236                                   ;输出：CX=描述符的选择子
237         push eax
238         push ebx
239         push edx
240
241         push ds
242         push es
243
244         mov ebx,core_data_seg_sel    ;切换到核心数据段
245         mov ds,ebx
246
247         sgdt [pgdt]                 ;以便开始处理GDT
248
249         mov ebx,mem_0_4_gb_seg_sel
250         mov es,ebx
251
252         movzx ebx,word [pgdt]        ;GDT界限
253         inc bx                       ;GDT总字节数，也是下一个描述符偏移
254         add ebx,[pgdt+2]             ;下一个描述符的线性地址
255
256         mov [es:ebx],eax
257         mov [es:ebx+4],edx
258
259         add word [pgdt],8            ;增加一个描述符的大小
260
261         lgdt [pgdt]                 ;对GDT的更改生效
262
263         mov ax,[pgdt]               ;得到GDT界限值
264         xor dx,dx
265         mov bx,8
266         div bx                      ;除以8，去掉余数
267         mov cx,ax
268         shl cx,3                    ;将索引号移到正确位置
269
270         pop es
271         pop ds
272
273         pop edx
274         pop ebx
275         pop eax
276

```

```

277         retf
278 ;-----
279 make_seg_descriptor:                ;构造存储器和系统的段描述符
280                                     ;输入: EAX=线性基地址
281                                     ;      EBX=段界限
282                                     ;      ECX=属性。各属性位都在原始
283                                     ;      位置, 无关的位清零
284                                     ;返回: EDX:EAX=描述符
285         mov edx,eax
286         shl eax,16
287         or ax,bx                    ;描述符前32位(EAX)构造完毕
288
289         and edx,0xffff0000          ;清除基地址中无关的位
290         rol edx,8
291         bswap edx                    ;装配基址的31~24和23~16 (80486+)
292
293         xor bx,bx
294         or edx,ebx                    ;装配段界限的高4位
295
296         or edx,ecx                    ;装配属性
297         retf
298
299 ;-----
300 make_gate_descriptor:                ;构造门的描述符(调用门等)
301                                     ;输入: EAX=门代码在段内偏移地址
302                                     ;      BX=门代码所在段的选择子
303                                     ;      CX=段类型及属性等(各属
304                                     ;      性位都在原始位置)
305                                     ;返回: EDX:EAX=完整的描述符
306
307         push ebx
308         push ecx
309
310         mov edx,eax
311         and edx,0xffff0000          ;得到偏移地址高16位
312         or dx,cx                    ;组装属性部分到EDX
313
314         and eax,0x0000ffff          ;得到偏移地址低16位
315         shl ebx,16
316         or eax,ebx                    ;组装段选择子部分
317
318         pop ecx
319         pop ebx
320
321         retf
322
323 ;-----
324 allocate_a_4k_page:                ;分配一个4KB的页
325                                     ;输入: 无
326                                     ;输出: EAX=页的物理地址
327
328         push ebx
329         push ecx
330         push edx
331         push ds
332
333         mov eax,core_data_seg_sel
334         mov ds,eax
335
336         xor eax,eax
337 .b1:
338         bts [page_bit_map],eax
339         jnc .b2
340         inc eax
341         cmp eax,page_map_len*8
342         jl .b1
343
344         mov ebx,message_3
345         call sys_routine_seg_sel:put_string
346         hlt                          ;没有可以分配的页, 停机
347
348 .b2:
349         shl eax,12                    ;乘以4096 (0x1000)
350
351         pop ds
352         pop edx
353         pop ecx
354         pop ebx
355
356         ret
357 ;-----
358 alloc_inst_a_page:                ;分配一个页, 并安装在当前活动的
359                                     ;层级分页结构中
360                                     ;输入: EBX=页的线性地址
361
362         push eax
363         push ebx
364         push esi
365         push ds
366
367         mov eax,mem_0_4_gb_seg_sel
368         mov ds,eax

```

```

369 ;检查该线性地址所对应的页表是否存在
370 mov esi,ebx
371 and esi,0xfffc0000
372 shr esi,20 ;得到页目录索引，并乘以4
373 or esi,0xfffff000 ;页目录自身的线性地址+表内偏移
374
375 test dword [esi],0x00000001 ;P位是否为“1”。检查该线性地址是
376 jnz .b1 ;否已经有对应的页表
377
378 ;创建该线性地址所对应的页表
379 call allocate_a_4k_page ;分配一个页做为页表
380 or eax,0x00000007
381 mov [esi],eax ;在页目录中登记该页表
382
383 .b1:
384 ;开始访问该线性地址所对应的页表
385 mov esi,ebx
386 shr esi,10
387 and esi,0x003ff000 ;或者0xfffff000，因高10位是零
388 or esi,0xffc00000 ;得到该页表的线性地址
389
390 ;得到该线性地址在页表内的对应条目（页表项）
391 and ebx,0x003ff000
392 shr ebx,10 ;相当于右移12位，再乘以4
393 or esi,ebx ;页表项的线性地址
394 call allocate_a_4k_page ;分配一个页，这才是要安装的页
395 or eax,0x00000007
396 mov [esi],eax
397
398 pop ds
399 pop esi
400 pop ebx
401 pop eax
402
403 retf
404
405 ;-----
406 create_copy_cur_pdir: ;创建新页目录，并复制当前页目录内容
407 ;输入：无
408 ;输出：EAX=新页目录的物理地址
409
410 push ds
411 push es
412 push esi
413 push edi
414 push ebx
415 push ecx
416
417 mov ebx,mem_0_4_gb_seg_sel
418 mov ds,ebx
419 mov es,ebx
420
421 call allocate_a_4k_page
422 mov ebx,eax
423 or ebx,0x00000007
424 mov [0xffffffff8],ebx
425
426 mov esi,0xffffffff000 ;ESI->当前页目录的线性地址
427 mov edi,0xfffffe000 ;EDI->新页目录的线性地址
428 mov ecx,1024 ;ECX=要复制的目录项数
429 cld
430 repe movsd
431
432 pop ecx
433 pop ebx
434 pop edi
435 pop esi
436 pop es
437 pop ds
438
439 retf
440
441 ;-----
442 terminate_current_task: ;终止当前任务
443 ;注意，执行此例程时，当前任务仍在
444 ;运行中。此例程其实也是当前任务的
445 ;一部分
446
447 mov eax,core_data_seg_sel
448 mov ds,eax
449
450 pushfd
451 pop edx
452
453 test dx,0100_0000_0000_0000B ;测试NT位
454 jnz .b1 ;当前任务是嵌套的，到.b1执行iretd
455 jmp far [program_man_tss] ;程序管理器任务
456
457 .b1:
458 iretd
459
460 sys_routine_end:
461 ;=====
462 SECTION core_data vstart=0 ;系统核心的数据段

```

```

461 ;-----
462         pgdt             dw 0             ;用于设置和修改GDT
463                         dd 0
464
465         page_bit_map     db 0xff,0xff,0xff,0xff,0xff,0x55,0x55,0xff
466                         db 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
467                         db 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
468                         db 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
469                         db 0x55,0x55,0x55,0x55,0x55,0x55,0x55,0x55
470                         db 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
471                         db 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
472                         db 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
473         page_map_len     equ $-page_bit_map
474
475 ;符号地址检索表
476 salt:
477 salt_1             db '@PrintString'
478                 times 256-($-salt_1) db 0
479                 dd put_string
480                 dw sys_routine_seg_sel
481
482 salt_2             db '@ReadDiskData'
483                 times 256-($-salt_2) db 0
484                 dd read_hard_disk_0
485                 dw sys_routine_seg_sel
486
487 salt_3             db '@PrintDwordAsHexString'
488                 times 256-($-salt_3) db 0
489                 dd put_hex_dword
490                 dw sys_routine_seg_sel
491
492 salt_4             db '@TerminateProgram'
493                 times 256-($-salt_4) db 0
494                 dd terminate_current_task
495                 dw sys_routine_seg_sel
496
497 salt_item_len      equ $-salt_4
498 salt_items         equ ($-salt)/salt_item_len
499
500 message_0          db ' Working in system core,protect mode.'
501                   db 0x0d,0x0a,0
502
503 message_1          db ' Paging is enabled.System core is mapped to'
504                   db ' address 0x80000000.',0x0d,0x0a,0
505
506 message_2          db 0x0d,0x0a
507                   db ' System wide CALL-GATE mounted.',0x0d,0x0a,0
508
509 message_3          db '*****No more pages*****',0
510
511 message_4          db 0x0d,0x0a,' Task switching...@ @',0x0d,0x0a,0
512
513 message_5          db 0x0d,0x0a,' Processor HALT.',0
514
515
516 bin_hex            db '0123456789ABCDEF'
517                   ;put_hex_dword子过程用的查找表
518
519 core_buf           times 512 db 0             ;内核用的缓冲区
520
521 cpu_brnd0          db 0x0d,0x0a,' ',0
522 cpu_brand          times 52 db 0
523 cpu_brnd1          db 0x0d,0x0a,0x0d,0x0a,0
524
525 ;任务控制块链
526 tcb_chain          dd 0
527
528 ;内核信息
529 core_next_laddr    dd 0x80100000             ;内核空间中下一个可分配的线性地址
530 program_man_tss     dd 0                     ;程序管理器的TSS描述符选择子
531                   dw 0
532
533 core_data_end:
534
535 ;=====
536 SECTION core_code vstart=0
537 ;-----
538 fill_descriptor_in_ldt: ;在LDT内安装一个新的描述符
539                       ;输入: EDX:EAX=描述符
540                       ; EBX=TCB基地址
541                       ;输出: CX=描述符的选择子
542
543         push eax
544         push edx
545         push edi
546         push ds
547
548         mov ecx,mem_0_4_gb_seg_sel
549         mov ds,ecx
550
551         mov edi,[ebx+0x0c] ;获得LDT基地址
552
553         xor ecx,ecx

```

```

553     mov cx,[ebx+0x0a]                ;获得LDT界限
554     inc cx                          ;LDT的总字节数，即新描述符偏移地址
555
556     mov [edi+ecx+0x00],eax
557     mov [edi+ecx+0x04],edx          ;安装描述符
558
559     add cx,8
560     dec cx                          ;得到新的LDT界限值
561
562     mov [ebx+0x0a],cx                ;更新LDT界限值到TCB
563
564     mov ax,cx
565     xor dx,dx
566     mov cx,8
567     div cx
568
569     mov cx,ax
570     shl cx,3                        ;左移3位，并且
571     or cx,0000_0000_0000_0100B     ;使TI位=1，指向LDT，最后使RPL=00
572
573     pop ds
574     pop edi
575     pop edx
576     pop eax
577
578     ret
579
580 ;-----
581 load_relocate_program:              ;加载并重定位用户程序
582                                     ;输入: PUSH 逻辑扇区号
583                                     ;      PUSH 任务控制块基地址
584                                     ;输出: 无
585
586     pushad
587
588     push ds
589     push es
590
591     mov ebp,esp                      ;为访问通过堆栈传递的参数做准备
592
593     mov ecx,mem_0_4_gb_seg_sel
594     mov es,ecx
595
596     ;清空当前页目录的前半部分（对应低2GB的局部地址空间）
597     mov ebx,0xffffffff000
598     xor esi,esi
599     .b1:
600     mov dword [es:ebx+esi*4],0x00000000
601     inc esi
602     cmp esi,512
603     jl .b1
604
605     ;以下开始分配内存并加载用户程序
606     mov eax,core_data_seg_sel
607     mov ds,eax                      ;切换DS到内核数据段
608
609     mov eax,[ebp+12*4]                ;从堆栈中取出用户程序起始扇区号
610     mov ebx,core_buf                 ;读取程序头部数据
611     call sys_routine_seg_sel:read_hard_disk_0
612
613     ;以下判断整个程序有多大
614     mov eax,[core_buf]                ;程序尺寸
615     mov ebx,eax
616     and ebx,0xffffffff000            ;使之4KB对齐
617     add ebx,0x1000
618     test eax,0x000000ffff            ;程序的大小正好是4KB的倍数吗?
619     cmovnz eax,ebx                    ;不是。使用凑整的结果
620
621     mov ecx,eax
622     shr ecx,12                       ;程序占用的总4KB页数
623
624     mov eax,mem_0_4_gb_seg_sel
625     mov ds,eax                      ;切换DS到0-4GB的段
626
627     mov eax,[ebp+12*4]                ;起始扇区号
628     mov esi,[ebp+11*4]                ;从堆栈中取得TCB的基地址
629     .b2:
630     mov ebx,[es:esi+0x06]            ;取得可用的线性地址
631     add dword [es:esi+0x06],0x1000
632     call sys_routine_seg_sel:alloc_inst_a_page
633
634     push ecx
635     mov ecx,8
636     .b3:
637     call sys_routine_seg_sel:read_hard_disk_0
638     inc eax
639     loop .b3
640
641     pop ecx
642     loop .b2
643
644     ;在内核地址空间内创建用户任务的TSS
645     mov eax,core_data_seg_sel        ;切换DS到内核数据段

```

```

645      mov ds, eax
646
647      mov ebx, [core_next_laddr]          ; 用户任务的TSS必须在全局空间上分配
648      call sys_routine_seg_sel:alloc_inst_a_page
649      add dword [core_next_laddr], 4096
650
651      mov [es:esi+0x14], ebx              ; 在TCB中填写TSS的线性地址
652      mov word [es:esi+0x12], 103         ; 在TCB中填写TSS的界限值
653
654      ; 在用户任务的局部地址空间内创建LDT
655      mov ebx, [es:esi+0x06]              ; 从TCB中取得可用的线性地址
656      add dword [es:esi+0x06], 0x1000
657      call sys_routine_seg_sel:alloc_inst_a_page
658      mov [es:esi+0x0c], ebx              ; 填写LDT线性地址到TCB中
659
660      ; 建立程序代码段描述符
661      mov eax, 0x00000000
662      mov ebx, 0x0000ffff
663      mov ecx, 0x00c0f800                ; 4KB粒度的代码段描述符, 特权级3
664      call sys_routine_seg_sel:make_seg_descriptor
665      mov ebx, esi                        ; TCB的基地址
666      call fill_descriptor_in_ldt
667      or cx, 0000_0000_0000_0011B        ; 设置选择子的特权级为3
668
669      mov ebx, [es:esi+0x14]              ; 从TCB中获取TSS的线性地址
670      mov [es:ebx+76], cx                 ; 填写TSS的CS域
671
672      ; 建立程序数据段描述符
673      mov eax, 0x00000000
674      mov ebx, 0x0000ffff
675      mov ecx, 0x00c0f200                ; 4KB粒度的数据段描述符, 特权级3
676      call sys_routine_seg_sel:make_seg_descriptor
677      mov ebx, esi                        ; TCB的基地址
678      call fill_descriptor_in_ldt
679      or cx, 0000_0000_0000_0011B        ; 设置选择子的特权级为3
680
681      mov ebx, [es:esi+0x14]              ; 从TCB中获取TSS的线性地址
682      mov [es:ebx+84], cx                 ; 填写TSS的DS域
683      mov [es:ebx+72], cx                 ; 填写TSS的ES域
684      mov [es:ebx+88], cx                 ; 填写TSS的FS域
685      mov [es:ebx+92], cx                 ; 填写TSS的GS域
686
687      ; 将数据段作为用户任务的3特权级固有堆栈
688      mov ebx, [es:esi+0x06]              ; 从TCB中取得可用的线性地址
689      add dword [es:esi+0x06], 0x1000
690      call sys_routine_seg_sel:alloc_inst_a_page
691
692      mov ebx, [es:esi+0x14]              ; 从TCB中获取TSS的线性地址
693      mov [es:ebx+80], cx                 ; 填写TSS的SS域
694      mov edx, [es:esi+0x06]              ; 堆栈的高端线性地址
695      mov [es:ebx+56], edx                 ; 填写TSS的ESP域
696
697      ; 在用户任务的局部地址空间内创建0特权级堆栈
698      mov ebx, [es:esi+0x06]              ; 从TCB中取得可用的线性地址
699      add dword [es:esi+0x06], 0x1000
700      call sys_routine_seg_sel:alloc_inst_a_page
701
702      mov eax, 0x00000000
703      mov ebx, 0x0000ffff
704      mov ecx, 0x00c09200                ; 4KB粒度的堆栈段描述符, 特权级0
705      call sys_routine_seg_sel:make_seg_descriptor
706      mov ebx, esi                        ; TCB的基地址
707      call fill_descriptor_in_ldt
708      or cx, 0000_0000_0000_0000B        ; 设置选择子的特权级为0
709
710      mov ebx, [es:esi+0x14]              ; 从TCB中获取TSS的线性地址
711      mov [es:ebx+8], cx                  ; 填写TSS的SS0域
712      mov edx, [es:esi+0x06]              ; 堆栈的高端线性地址
713      mov [es:ebx+4], edx                 ; 填写TSS的ESP0域
714
715      ; 在用户任务的局部地址空间内创建1特权级堆栈
716      mov ebx, [es:esi+0x06]              ; 从TCB中取得可用的线性地址
717      add dword [es:esi+0x06], 0x1000
718      call sys_routine_seg_sel:alloc_inst_a_page
719
720      mov eax, 0x00000000
721      mov ebx, 0x0000ffff
722      mov ecx, 0x00c0b200                ; 4KB粒度的堆栈段描述符, 特权级1
723      call sys_routine_seg_sel:make_seg_descriptor
724      mov ebx, esi                        ; TCB的基地址
725      call fill_descriptor_in_ldt
726      or cx, 0000_0000_0000_0001B        ; 设置选择子的特权级为1
727
728      mov ebx, [es:esi+0x14]              ; 从TCB中获取TSS的线性地址
729      mov [es:ebx+16], cx                 ; 填写TSS的SS1域
730      mov edx, [es:esi+0x06]              ; 堆栈的高端线性地址
731      mov [es:ebx+12], edx                 ; 填写TSS的ESP1域
732
733      ; 在用户任务的局部地址空间内创建2特权级堆栈
734      mov ebx, [es:esi+0x06]              ; 从TCB中取得可用的线性地址
735      add dword [es:esi+0x06], 0x1000
736      call sys_routine_seg_sel:alloc_inst_a_page

```

```

737
738     mov eax,0x00000000
739     mov ebx,0x000fffff
740     mov ecx,0x00c0d200                ;4KB粒度的堆栈段描述符，特权级2
741     call sys_routine_seg_sel:make_seg_descriptor
742     mov ebx,esi                        ;TCB的基地址
743     call fill_descriptor_in_ldt
744     or cx,0000_0000_0000_0010B        ;设置选择子的特权级为2
745
746     mov ebx,[es:esi+0x14]              ;从TCB中获取TSS的线性地址
747     mov [es:ebx+24],cx                ;填写TSS的SS2域
748     mov edx,[es:esi+0x06]              ;堆栈的高端线性地址
749     mov [es:ebx+20],edx                ;填写TSS的ESP2域
750
751
752     ;重定位SALT
753     mov eax,mem_0_4_gb_seg_sel        ;访问任务的4GB虚拟地址空间时用
754     mov es,eax
755
756     mov eax,core_data_seg_sel
757     mov ds,eax
758
759     cld
760
761     mov ecx,[es:0x0c]                  ;U-SALT条目数
762     mov edi,[es:0x08]                  ;U-SALT在4GB空间内的偏移
763
764     .b4:
765     push ecx
766     push edi
767
768     mov ecx,salt_items
769     mov esi,salt
770
771     .b5:
772     push edi
773     push esi
774     push ecx
775
776     mov ecx,64                        ;检索表中，每条目的比较次数
777     repe cmpsd                        ;每次比较4字节
778     jnz .b6
779     mov eax,[esi]                     ;若匹配，则esi恰好指向其后的地址
780     mov [es:edi-256],eax               ;将字符串改写成偏移地址
781     mov ax,[esi+4]
782     or ax,0000000000000011B           ;以用户程序自己的特权级使用调用门
783     mov [es:edi-252],ax                ;故RPL=3
784     ;回填调用门选择子
785     .b6:
786
787     pop ecx
788     pop esi
789     add esi,salt_item_len
790     pop edi
791     loop .b5
792
793     pop edi
794     add edi,256
795     pop ecx
796     loop .b4
797
798     ;在GDT中登记LDT描述符
799     mov esi,[ebp+11*4]                 ;从堆栈中取得TCB的基地址
800     mov eax,[es:esi+0x0c]              ;LDT的起始线性地址
801     movzx ebx,word [es:esi+0x0a]        ;LDT段界限
802     mov ecx,0x00408200                 ;LDT描述符，特权级0
803     call sys_routine_seg_sel:make_seg_descriptor
804     call sys_routine_seg_sel:set_up_gdt_descriptor
805     mov [es:esi+0x10],cx                ;登记LDT选择子到TCB中
806
807     mov ebx,[es:esi+0x14]              ;从TCB中获取TSS的线性地址
808     mov [es:ebx+96],cx                 ;填写TSS的LDT域
809
810     mov word [es:ebx+0],0               ;反向链=0
811
812     mov dx,[es:esi+0x12]                ;段长度（界限）
813     mov [es:ebx+102],dx                 ;填写TSS的I/O位图偏移域
814
815     mov word [es:ebx+100],0             ;T=0
816
817     mov eax,[es:0x04]                  ;从任务的4GB地址空间获取入口点
818     mov [es:ebx+32],eax                 ;填写TSS的EIP域
819
820     pushfd
821     pop edx
822     mov [es:ebx+36],edx                 ;填写TSS的EFLAGS域
823
824     ;在GDT中登记TSS描述符
825     mov eax,[es:esi+0x14]              ;从TCB中获取TSS的起始线性地址
826     movzx ebx,word [es:esi+0x12]        ;段长度（界限）
827     mov ecx,0x00408900                 ;TSS描述符，特权级0
828     call sys_routine_seg_sel:make_seg_descriptor
829     call sys_routine_seg_sel:set_up_gdt_descriptor
830     mov [es:esi+0x18],cx                ;登记TSS选择子到TCB

```

```

829
830 ;创建用户任务的项目录
831 ;注意！页的分配和使用是由页位图决定的，可以不占用线性地址空间
832 call sys_routine_seg_sel:create_copy_cur_pdir
833 mov ebx,[es:esi+0x14] ;从TCB中获取TSS的线性地址
834 mov dword [es:ebx+28],eax ;填写TSS的CR3(PDBR)域
835
836 pop es ;恢复到调用此过程前的es段
837 pop ds ;恢复到调用此过程前的ds段
838
839 popad
840
841 ret 8 ;丢弃调用本过程前压入的参数
842
843 ;-----
844 append_to_tcb_link: ;在TCB链上追加任务控制块
845 ;输入：ECX=TCB线性基地址
846 push eax
847 push edx
848 push ds
849 push es
850
851 mov eax,core_data_seg_sel ;令DS指向内核数据段
852 mov ds,eax
853 mov eax,mem_0_4_gb_seg_sel ;令ES指向0..4GB段
854 mov es,eax
855
856 mov dword [es: ecx+0x00],0 ;当前TCB指针域清零，以指示这是最
857 ;后一个TCB
858
859 mov eax,[tcb_chain] ;TCB表头指针
860 or eax,eax ;链表为空？
861 jz .notcb
862
863 .searc:
864 mov edx,eax
865 mov eax,[es: edx+0x00]
866 or eax,eax
867 jnz .searc
868
869 mov [es: edx+0x00],ecx
870 jmp .retpc
871
872 .notcb:
873 mov [tcb_chain],ecx ;若为空表，直接令表头指针指向TCB
874
875 .retpc:
876 pop es
877 pop ds
878 pop edx
879 pop eax
880
881 ret
882
883 ;-----
884 start:
885 mov ecx,core_data_seg_sel ;令DS指向核心数据段
886 mov ds,ecx
887
888 mov ecx,mem_0_4_gb_seg_sel ;令ES指向4GB数据段
889 mov es,ecx
890
891 mov ebx,message_0
892 call sys_routine_seg_sel:put_string
893
894 ;显示处理器品牌信息
895 mov eax,0x80000002
896 cpuid
897 mov [cpu_brand + 0x00],eax
898 mov [cpu_brand + 0x04],ebx
899 mov [cpu_brand + 0x08],ecx
900 mov [cpu_brand + 0x0c],edx
901
902 mov eax,0x80000003
903 cpuid
904 mov [cpu_brand + 0x10],eax
905 mov [cpu_brand + 0x14],ebx
906 mov [cpu_brand + 0x18],ecx
907 mov [cpu_brand + 0x1c],edx
908
909 mov eax,0x80000004
910 cpuid
911 mov [cpu_brand + 0x20],eax
912 mov [cpu_brand + 0x24],ebx
913 mov [cpu_brand + 0x28],ecx
914 mov [cpu_brand + 0x2c],edx
915
916 mov ebx,cpu_brnd0 ;显示处理器品牌信息
917 call sys_routine_seg_sel:put_string
918 mov ebx,cpu_brand
919 call sys_routine_seg_sel:put_string
920 mov ebx,cpu_brnd1

```



```

921      call sys_routine_seg_sel:put_string
922
923      ;准备打开分页机制
924
925      ;创建系统内核的页目录表PDT
926      ;页目录表清零
927      mov ecx,1024                ;1024个目录项
928      mov ebx,0x00020000          ;页目录的物理地址
929      xor esi,esi
930
931      .b1:
932      mov dword [es:ebx+esi],0x00000000 ;页目录表项清零
933      add esi,4
934      loop .b1
935
936      ;在页目录内创建指向页目录自己的目录项
937      mov dword [es:ebx+4092],0x00020003
938
939      ;在页目录内创建与线性地址0x00000000对应的目录项
940      mov dword [es:ebx+0],0x00021003 ;写入目录项（页表的物理地址和属性）
941
942      ;创建与上面那个目录项相对应的页表，初始化页表项
943      mov ebx,0x00021000            ;页表的物理地址
944      xor eax,eax                    ;起始页的物理地址
945      xor esi,esi
946
947      .b2:
948      mov edx,eax
949      or edx,0x00000003
950      mov [es:ebx+esi*4],edx         ;登记页的物理地址
951      add eax,0x1000                ;下一个相邻页的物理地址
952      inc esi
953      cmp esi,256                    ;仅低端1MB内存对应的页才是有效的
954      jl .b2
955
956      .b3:
957      ;其余的页表项置为无效
958      mov dword [es:ebx+esi*4],0x00000000
959      inc esi
960      cmp esi,1024
961      jl .b3
962
963      ;令CR3寄存器指向页目录，并正式开启页功能
964      mov eax,0x00020000            ;PCD=PWT=0
965      mov cr3,eax
966
967      mov eax,cr0
968      or eax,0x80000000
969      mov cr0,eax                    ;开启分页机制
970
971      ;在页目录内创建与线性地址0x80000000对应的目录项
972      mov ebx,0xffffffff             ;页目录自己的线性地址
973      mov esi,0x80000000             ;映射的起始地址
974      shr esi,22                      ;线性地址的高10位是目录索引
975      shl esi,2
976      mov dword [es:ebx+esi],0x00021003 ;写入目录项（页表的物理地址和属性）
977      ;目标单元的线性地址为0xFFFFF200
978
979      ;将GDT中的段描述符映射到线性地址0x80000000
980      sgdt [pgdt]
981
982      mov ebx,[pgdt+2]
983
984      or dword [es:ebx+0x10+4],0x80000000
985      or dword [es:ebx+0x18+4],0x80000000
986      or dword [es:ebx+0x20+4],0x80000000
987      or dword [es:ebx+0x28+4],0x80000000
988      or dword [es:ebx+0x30+4],0x80000000
989      or dword [es:ebx+0x38+4],0x80000000
990
991      add dword [pgdt+2],0x80000000 ;GDTR也用的是线性地址
992
993      lgdt [pgdt]
994
995      jmp core_code_seg_sel:flush    ;刷新段寄存器CS，启用高端线性地址
996
997      flush:
998      mov eax,core_stack_seg_sel
999      mov ss,eax
1000
1001      mov eax,core_data_seg_sel
1002      mov ds,eax
1003
1004      mov ebx,message_1
1005      call sys_routine_seg_sel:put_string
1006
1007      ;以下开始安装为整个系统服务的调用门。特权级之间的控制转移必须使用门
1008      mov edi,salt                  ;C-SALT表的起始位置
1009      mov ecx,salt_items            ;C-SALT表的条目数量
1010
1011      .b4:
1012      push ecx
1013      mov eax,[edi+256]              ;该条目入口点的32位偏移地址
1014      mov bx,[edi+260]              ;该条目入口点的段选择子
1015      mov cx,1_11_0_1100_000_00000B ;特权级3的调用门(3以上的特权级才
1016      ;允许访问)，0个参数(因为用寄存器

```

```

1013                                     ;传递参数, 而没有用栈)
1014 call sys_routine_seg_sel:make_gate_descriptor
1015 call sys_routine_seg_sel:set_up_gdt_descriptor
1016 mov [edi+260],cx                     ;将返回的门描述符选择子回填
1017 add edi,salt_item_len                ;指向下一个C-SALT条目
1018 pop ecx
1019 loop .b4
1020
1021 ;对门进行测试
1022 mov ebx,message_2
1023 call far [salt_1+256]                ;通过门显示信息(偏移量将被忽略)
1024
1025 ;为程序管理器的TSS分配内存空间
1026 mov ebx,[core_next_laddr]
1027 call sys_routine_seg_sel:alloc_inst_a_page
1028 add dword [core_next_laddr],4096
1029
1030 ;在程序管理器的TSS中设置必要的项目
1031 mov word [es:ebx+0],0                ;反向链=0
1032
1033 mov eax,cr3
1034 mov dword [es:ebx+28],eax            ;登记CR3(PDBR)
1035
1036 mov word [es:ebx+96],0                ;没有LDT。处理器允许没有LDT的任务。
1037 mov word [es:ebx+100],0              ;T=0
1038 mov word [es:ebx+102],103            ;没有I/O位图。0特权级事实上不需要。
1039
1040 ;创建程序管理器的TSS描述符, 并安装到GDT中
1041 mov eax,ebx                          ;TSS的起始线性地址
1042 mov ebx,103                          ;段长度(界限)
1043 mov ecx,0x00408900                  ;TSS描述符, 特权级0
1044 call sys_routine_seg_sel:make_seg_descriptor
1045 call sys_routine_seg_sel:set_up_gdt_descriptor
1046 mov [program_man_tss+4],cx           ;保存程序管理器的TSS描述符选择子
1047
1048 ;任务寄存器TR中的内容是任务存在的标志, 该内容也决定了当前任务是谁。
1049 ;下面的指令为当前正在执行的0特权级任务“程序管理器”后补手续(TSS)。
1050 ltr cx
1051
1052 ;现在可认为“程序管理器”任务正执行中
1053
1054 ;创建用户任务的任务控制块
1055 mov ebx,[core_next_laddr]
1056 call sys_routine_seg_sel:alloc_inst_a_page
1057 add dword [core_next_laddr],4096
1058
1059 mov dword [es:ebx+0x06],0             ;用户任务局部空间的分配从0开始。
1060 mov word [es:ebx+0x0a],0xffff         ;登记LDT初始的界限到TCB中
1061 mov ecx,ebx
1062 call append_to_tcb_link               ;将此TCB添加到TCB链中
1063
1064 push dword 50                         ;用户程序位于逻辑50扇区
1065 push ecx                              ;压入任务控制块起始线性地址
1066
1067 call load_relocate_program
1068
1069 mov ebx,message_4
1070 call sys_routine_seg_sel:put_string
1071
1072 call far [es:ecx+0x14]                ;执行任务切换。
1073
1074 mov ebx,message_5
1075 call sys_routine_seg_sel:put_string
1076
1077 hlt
1078
1079 core_code_end:
1080
1081 ;-----
1082 SECTION core_trail
1083 ;-----
1084 core_end:

```

```

1      ;代码清单17-2
2      ;文件名: c17_core.asm
3      ;文件说明: 保护模式微型核心程序
4      ;创建日期: 2012-07-12 23:15
5      ;-----
6      ;以下定义常量
7      flat_4gb_code_seg_sel equ 0x0008      ;平坦模型下的4GB代码段选择子
8      flat_4gb_data_seg_sel equ 0x0018      ;平坦模型下的4GB数据段选择子
9      idt_linear_address    equ 0x8001f000   ;中断描述符表的线性基地址
10     ;-----
11     ;以下定义宏
12     %macro alloc_core_linear 0              ;在内核空间中分配虚拟内存
13         mov ebx,[core_tcb+0x06]
14         add dword [core_tcb+0x06],0x1000
15         call flat_4gb_code_seg_sel:alloc_inst_a_page
16     %endmacro
17     ;-----
18     %macro alloc_user_linear 0              ;在任务空间中分配虚拟内存
19         mov ebx,[esi+0x06]
20         add dword [esi+0x06],0x1000
21         call flat_4gb_code_seg_sel:alloc_inst_a_page
22     %endmacro
23
24     ;=====
25     SECTION core vstart=0x80040000
26
27     ;以下是系统核心的头部, 用于加载核心程序
28     core_length      dd core_end            ;核心程序总长度#00
29
30     core_entry       dd start              ;核心代码段入口点#04
31
32     ;-----
33     [bits 32]
34     ;-----
35     ;字符串显示例程(适用于平坦内存模型)
36     put_string:      ;显示0终止的字符串并移动光标
37                     ;输入: EBX=字符串的线性地址
38
39         push ebx
40         push ecx
41
42         cli          ;硬件操作期间, 关中断
43
44     .getc:
45         mov cl,[ebx]
46         or cl,cl      ;检测串结束标志(0)
47         jz .exit      ;显示完毕, 返回
48         call put_char
49         inc ebx
50         jmp .getc
51
52     .exit:
53
54         sti          ;硬件操作完毕, 开放中断
55
56         pop ecx
57         pop ebx
58
59         retf         ;段间返回
60
61     ;-----
62     put_char:        ;在当前光标处显示一个字符,并推进
63                     ;光标。仅用于段内调用
64                     ;输入: CL=字符ASCII码
65
66         pushad
67
68         ;以下取当前光标位置
69         mov dx,0x3d4
70         mov al,0x0e
71         out dx,al
72         inc dx          ;0x3d5
73         in al,dx        ;高字
74         mov ah,al
75
76         dec dx          ;0x3d4
77         mov al,0x0f
78         out dx,al
79         inc dx          ;0x3d5
80         in al,dx        ;低字
81         mov bx,ax       ;BX=代表光标位置的16位数
82         and ebx,0x0000ffff ;准备使用32位寻址方式访问显存
83
84         cmp cl,0x0d      ;回车符?
85         jnz .put_0a
86
87         mov ax,bx        ;以下按回车符处理
88         mov bl,80
89         div bl
90         mul bl
91         mov bx,ax
92         jmp .set_cursor

```

```

93     .put_0a:
94         cmp cl,0x0a                ;换行符?
95         jnz .put_other
96         add bx,80                  ;增加一行
97         jmp .roll_screen
98
99     .put_other:                    ;正常显示字符
100        shl bx,1
101        mov [0x800b8000+ebx],cl    ;在光标位置处显示字符
102
103        ;以下将光标位置推进一个字符
104        shr bx,1
105        inc bx
106
107     .roll_screen:
108        cmp bx,2000                ;光标超出屏幕? 滚屏
109        jl .set_cursor
110
111        cld
112        mov esi,0x800b80a0         ;小心! 32位模式下movsb/w/d
113        mov edi,0x800b8000         ;使用的是esi/edi/ecx
114        mov ecx,1920
115        rep movsd
116        mov bx,3840                ;清除屏幕最底一行
117        mov ecx,80                 ;32位程序应该使用ECX
118
119     .cls:
120        mov word [0x800b8000+ebx],0x0720
121        add bx,2
122        loop .cls
123
124        mov bx,1920
125
126     .set_cursor:
127        mov dx,0x3d4
128        mov al,0x0e
129        out dx,al
130        inc dx                      ;0x3d5
131        mov al,bh
132        out dx,al
133        dec dx                      ;0x3d4
134        mov al,0x0f
135        out dx,al
136        inc dx                      ;0x3d5
137        mov al,b1
138        out dx,al
139
140        popad
141        ret
142
143 ;-----
144 read_hard_disk_0:                ;从硬盘读取一个逻辑扇区 (平坦模型)
145                                ;EAX=逻辑扇区号
146                                ;EBX=目标缓冲区线性地址
147                                ;返回: EBX=EBX+512
148
149        cli
150
151        push eax
152        push ecx
153        push edx
154
155        push eax
156        mov dx,0x1f2
157        mov al,1
158        out dx,al                  ;读取的扇区数
159
160        inc dx                      ;0x1f3
161        pop eax
162        out dx,al                  ;LBA地址7~0
163
164        inc dx                      ;0x1f4
165        mov cl,8
166        shr eax,cl
167        out dx,al                  ;LBA地址15~8
168
169        inc dx                      ;0x1f5
170        shr eax,cl
171        out dx,al                  ;LBA地址23~16
172
173        inc dx                      ;0x1f6
174        shr eax,cl
175        or al,0xe0                 ;第一硬盘 LBA地址27~24
176        out dx,al
177
178        inc dx                      ;0x1f7
179        mov al,0x20                 ;读命令
180        out dx,al
181
182     .waits:
183        in al,dx
184        and al,0x88

```

```

185         cmp al,0x08
186         jnz .waits           ;不忙，且硬盘已准备好数据传输
187
188         mov ecx,256          ;总共要读取的字数
189         mov dx,0x1f0
190     .readw:
191         in ax,dx
192         mov [ebx],ax
193         add ebx,2
194         loop .readw
195
196         pop edx
197         pop ecx
198         pop eax
199
200         sti
201
202         retf                 ;远返回
203
204 ;-----
205 ;汇编语言程序是极难一次成功，而且调试非常困难。这个例程可以提供帮助
206 put_hex_dword:               ;在当前光标处以十六进制形式显示
207                             ;一个双字并推进光标
208                             ;输入：EDX=要转换并显示的数字
209                             ;输出：无
210
211         pushad
212
213         mov ebx,bin_hex      ;指向核心地址空间内的转换表
214         mov ecx,8
215     .xlt:
216         rol edx,4
217         mov eax,edx
218         and eax,0x0000000f
219         xlat
220
221         push ecx
222         mov cl,al
223         call put_char
224         pop ecx
225
226         loop .xlt
227
228         popad
229         retf
230 ;-----
231 set_up_gdt_descriptor:      ;在GDT内安装一个新的描述符
232                             ;输入：EDX:EAX=描述符
233                             ;输出：CX=描述符的选择子
234
235         push eax
236         push ebx
237         push edx
238
239         sgdt [pgdt]          ;取得GDTR的界限和线性地址
240
241         movzx ebx,word [pgdt] ;GDT界限
242         inc bx                ;GDT总字节数，也是下一个描述符偏移
243         add ebx,[pgdt+2]      ;下一个描述符的线性地址
244
245         mov [ebx],eax
246         mov [ebx+4],edx
247
248         add word [pgdt],8     ;增加一个描述符的大小
249
250         lgdt [pgdt]          ;对GDT的更改生效
251
252         mov ax,[pgdt]         ;得到GDT界限值
253         xor dx,dx
254         mov bx,8
255         div bx                ;除以8，去掉余数
256         mov cx,ax
257         shl cx,3              ;将索引号移到正确位置
258
259         pop edx
260         pop ebx
261         pop eax
262
263         retf
264 ;-----
265 make_seg_descriptor:        ;构造存储器和系统的段描述符
266                             ;输入：EAX=线性基地址
267                             ;      EBX=段界限
268                             ;      ECX=属性。各属性位都在原始
269                             ;      位置，无关的位清零
270                             ;返回：EDX:EAX=描述符
271
272         mov edx,eax
273         shl eax,16
274         or ax,bx              ;描述符前32位(EAX)构造完毕
275
276         and edx,0xffff0000    ;清除基地址中无关的位
277         rol edx,8
278         bswap edx             ;装配基址的31~24和23~16 (80486+)

```

```

277
278     xor bx,bx
279     or  edx,ebx                ;装配段界限的高4位
280
281     or  edx,ecx                ;装配属性
282
283     retf
284
285 ;-----
286 make_gate_descriptor:         ;构造门的描述符（调用门等）
287                               ;输入： EAX=门代码在段内偏移地址
288                               ;      BX=门代码所在段的选择子
289                               ;      CX=段类型及属性等（各属
290                               ;      性位都在原始位置）
291                               ;返回： EDX:EAX=完整的描述符
292     push ebx
293     push ecx
294
295     mov edx,eax
296     and edx,0xffff0000        ;得到偏移地址高16位
297     or  dx,cx                 ;组装属性部分到EDX
298
299     and eax,0x0000ffff        ;得到偏移地址低16位
300     shl ebx,16
301     or  eax,ebx                ;组装段选择子部分
302
303     pop ecx
304     pop ebx
305
306     retf
307
308 ;-----
309 allocate_a_4k_page:           ;分配一个4KB的页
310                               ;输入： 无
311                               ;输出： EAX=页的物理地址
312     push ebx
313     push ecx
314     push edx
315
316     xor eax,eax
317 .b1:
318     bts [page_bit_map],eax
319     jnc .b2
320     inc eax
321     cmp eax,page_map_len*8
322     jl  .b1
323
324     mov ebx,message_3
325     call flat_4gb_code_seg_sel:put_string
326     hlt                        ;没有可以分配的页， 停机
327
328 .b2:
329     shl eax,12                 ;乘以4096（0x1000）
330
331     pop edx
332     pop ecx
333     pop ebx
334
335     ret
336
337 ;-----
338 alloc_inst_a_page:            ;分配一个页，并安装在当前活动的
339                               ;层级分页结构中
340                               ;输入： EBX=页的线性地址
341     push eax
342     push ebx
343     push esi
344
345     ;检查该线性地址所对应的页表是否存在
346     mov esi,ebx
347     and esi,0xffc00000
348     shr esi,20                 ;得到页目录索引，并乘以4
349     or  esi,0xfffff000        ;页目录自身的线性地址+表内偏移
350
351     test dword [esi],0x00000001 ;P位是否为“1”。检查该线性地址是
352     jnz .b1                    ;否已经有对应的页表
353
354     ;创建该线性地址所对应的页表
355     call allocate_a_4k_page    ;分配一个页做为页表
356     or  eax,0x00000007
357     mov [esi],eax             ;在页目录中登记该页表
358
359 .b1:
360     ;开始访问该线性地址所对应的页表
361     mov esi,ebx
362     shr esi,10
363     and esi,0x003ff000        ;或者0xfffff000，因高10位是零
364     or  esi,0xffc00000        ;得到该页表的线性地址
365
366     ;得到该线性地址在页表内的对应条目（页表项）
367     and ebx,0x003ff000
368     shr ebx,10                 ;相当于右移12位，再乘以4

```

```

369         or esi,ebx                ;页表项的线性地址
370         call allocate_a_4k_page    ;分配一个页，这才是要安装的页
371         or eax,0x00000007
372         mov [esi],eax
373
374         pop esi
375         pop ebx
376         pop eax
377
378         retf
379
380 ;-----
381 create_copy_cur_pdir:              ;创建新页目录，并复制当前页目录内容
382                                     ;输入：无
383                                     ;输出：EAX=新页目录的物理地址
384         push esi
385         push edi
386         push ebx
387         push ecx
388
389         call allocate_a_4k_page
390         mov ebx,eax
391         or ebx,0x00000007
392         mov [0xffffffff8],ebx
393
394         invlpg [0xffffffff8]
395
396         mov esi,0xfffff000          ;ESI->当前页目录的线性地址
397         mov edi,0xfffffe000         ;EDI->新页目录的线性地址
398         mov ecx,1024                ;ECX=要复制的目录项数
399         cld
400         repe movsd
401
402         pop ecx
403         pop ebx
404         pop edi
405         pop esi
406
407         retf
408
409 ;-----
410 general_interrupt_handler:          ;通用的中断处理过程
411         push eax
412
413         mov al,0x20                 ;中断结束命令EOI
414         out 0xa0,al                 ;向从片发送
415         out 0x20,al                 ;向主片发送
416
417         pop eax
418
419         iretd
420
421 ;-----
422 general_exception_handler:          ;通用的异常处理过程
423         mov ebx,excep_msg
424         call flat_4gb_code_seg_sel:put_string
425
426         hlt
427
428 ;-----
429 rtm_0x70_interrupt_handle:          ;实时时钟中断处理过程
430
431         pushad
432
433         mov al,0x20                 ;中断结束命令EOI
434         out 0xa0,al                 ;向8259A从片发送
435         out 0x20,al                 ;向8259A主片发送
436
437         mov al,0x0c                 ;寄存器C的索引。且开放NMI
438         out 0x70,al
439         in al,0x71                  ;读一下RTC的寄存器C，否则只发生一次中断
440                                     ;此处不考虑闹钟和周期性中断的情况
441         ;找当前任务（状态为忙的任务）在链表中的位置
442         mov eax,tcb_chain
443         .b0:                         ;EAX=链表头或当前TCB线性地址
444         mov ebx,[eax]               ;EBX=下一个TCB线性地址
445         or ebx,ebx
446         jz .irtn                    ;链表为空，或已到末尾，从中断返回
447         cmp word [ebx+0x04],0xffff  ;是忙任务（当前任务）？
448         je .b1
449         mov eax,ebx                 ;定位到下一个TCB（的线性地址）
450         jmp .b0
451
452         ;将当前为忙的任务移到链尾
453         .b1:
454         mov ecx,[ebx]               ;下游TCB的线性地址
455         mov [eax],ecx              ;将当前任务从链中拆除
456
457         .b2:                         ;此时，EBX=当前任务的线性地址
458         mov edx,[eax]
459         or edx,edx                  ;已到链表尾端？
460         jz .b3

```

```

461      mov eax,edx
462      jmp .b2
463
464      .b3:
465      mov [eax],ebx          ;将忙任务的TCB挂在链表尾端
466      mov dword [ebx],0x00000000 ;将忙任务的TCB标记为链尾
467
468      ;从链首搜索第一个空闲任务
469      mov eax,tcb_chain
470      .b4:
471      mov eax,[eax]
472      or eax,eax             ;已到链尾（未发现空闲任务）
473      jz .irtn               ;未发现空闲任务，从中断返回
474      cmp word [eax+0x04],0x0000 ;是空闲任务？
475      jnz .b4
476
477      ;将空闲任务和当前任务的状态都取反
478      not word [eax+0x04]     ;设置空闲任务的状态为忙
479      not word [ebx+0x04]     ;设置当前任务（忙）的状态为空闲
480      jmp far [eax+0x14]      ;任务转换
481
482      .irtn:
483      popad
484
485      iretd
486
487      ;-----
488      terminate_current_task: ;终止当前任务
489                              ;注意，执行此例程时，当前任务仍在
490                              ;运行中。此例程其实也是当前任务的
491                              ;一部分
492      ;找当前任务（状态为忙的任务）在链表中的位置
493      mov eax,tcb_chain
494      .b0:
495      mov ebx,[eax]          ;EAX=链表头或当前TCB线性地址
496      cmp word [ebx+0x04],0xffff ;EBX=下一个TCB线性地址
497      je .b1                 ;是忙任务（当前任务）？
498      mov eax,ebx            ;定位到下一个TCB（的线性地址）
499      jmp .b0
500
501      .b1:
502      mov word [ebx+0x04],0x3333 ;修改当前任务的状态为“退出”
503
504      .b2:
505      hlt                   ;停机，等待程序管理器恢复运行时，
506                              ;将其回收
507      jmp .b2
508
509      ;-----
510      pgdt      dw 0          ;用于设置和修改GDT
511               dd 0
512
513      pidt      dw 0
514               dd 0
515
516      ;任务控制块链
517      tcb_chain  dd 0
518
519      core_tcb   times 32 db 0 ;内核（程序管理器）的TCB
520
521      page_bit_map db 0xff,0xff,0xff,0xff,0xff,0xff,0x55,0x55
522                  db 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
523                  db 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
524                  db 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
525                  db 0x55,0x55,0x55,0x55,0x55,0x55,0x55,0x55
526                  db 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
527                  db 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
528                  db 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
529      page_map_len equ $-page_bit_map
530
531      ;符号地址检索表
532      salt:
533      salt_1     db '@PrintString'
534                  times 256-($-salt_1) db 0
535                  dd put_string
536                  dw flat_4gb_code_seg_sel
537
538      salt_2     db '@ReadDiskData'
539                  times 256-($-salt_2) db 0
540                  dd read_hard_disk_0
541                  dw flat_4gb_code_seg_sel
542
543      salt_3     db '@PrintDwordAsHexString'
544                  times 256-($-salt_3) db 0
545                  dd put_hex_dword
546                  dw flat_4gb_code_seg_sel
547
548      salt_4     db '@TerminateProgram'
549                  times 256-($-salt_4) db 0
550                  dd terminate_current_task
551                  dw flat_4gb_code_seg_sel
552

```



```

553     salt_item_len    equ $-salt_4
554     salt_items       equ ($-salt)/salt_item_len
555
556     excep_msg        db  '*****Exception encounted*****',0
557
558     message_0         db  ' Working in system core with protection '
559                       db  'and paging are all enabled.System core is mapped '
560                       db  'to address 0x80000000.',0x0d,0x0a,0
561
562     message_1         db  ' System wide CALL-GATE mounted.',0x0d,0x0a,0
563
564     message_3         db  '*****No more pages*****',0
565
566     core_msg0         db  ' System core task running!',0x0d,0x0a,0
567
568     bin_hex           db  '0123456789ABCDEF'
569                       ;put_hex_dword子过程用的查找表
570
571     core_buf          times 512 db 0                ;内核用的缓冲区
572
573     cpu_brnd0         db  0x0d,0x0a,' ',0
574     cpu_brand         times 52 db 0
575     cpu_brnd1         db  0x0d,0x0a,0x0d,0x0a,0
576
577 ;-----
578 fill_descriptor_in_ldt:                ;在LDT内安装一个新的描述符
579                                       ;输入: EDX:EAX=描述符
580                                       ;          EBX=TCB基地址
581                                       ;输出: CX=描述符的选择子
582     push eax
583     push edx
584     push edi
585
586     mov edi,[ebx+0x0c]                ;获得LDT基地址
587
588     xor ecx,ecx
589     mov cx,[ebx+0x0a]                ;获得LDT界限
590     inc cx                            ;LDT的总字节数, 即新描述符偏移地址
591
592     mov [edi+ecx+0x00],eax
593     mov [edi+ecx+0x04],edx            ;安装描述符
594
595     add cx,8
596     dec cx                            ;得到新的LDT界限值
597
598     mov [ebx+0x0a],cx                ;更新LDT界限值到TCB
599
600     mov ax,cx
601     xor dx,dx
602     mov cx,8
603     div cx
604
605     mov cx,ax
606     shl cx,3                        ;左移3位, 并且
607     or cx,0000_0000_0000_0100B      ;使TI位=1, 指向LDT, 最后使RPL=00
608
609     pop edi
610     pop edx
611     pop eax
612
613     ret
614
615 ;-----
616 load_relocate_program:                ;加载并重定位用户程序
617                                       ;输入: PUSH 逻辑扇区号
618                                       ;          PUSH 任务控制块基地址
619                                       ;输出: 无
620
621     pushad
622
623     mov ebp,esp                      ;为访问通过堆栈传递的参数做准备
624
625     ;清空当前页目录的前半部分 (对应低2GB的局部地址空间)
626     mov ebx,0xffffffff000
627     xor esi,esi
628     .b1:
629     mov dword [ebx+esi*4],0x00000000
630     inc esi
631     cmp esi,512
632     jl .b1
633
634     mov eax,cr3
635     mov cr3,eax                      ;刷新TLB
636
637     ;以下开始分配内存并加载用户程序
638     mov eax,[ebp+40]                 ;从堆栈中取出用户程序起始扇区号
639     mov ebx,core_buf                 ;读取程序头部数据
640     call flat_4gb_code_seg_sel:read_hard_disk_0
641
642     ;以下判断整个程序有多大
643     mov eax,[core_buf]                ;程序尺寸
644     mov ebx,eax
645     and ebx,0xffffffff000            ;使之4KB对齐

```

```

645      add ebx,0x1000
646      test eax,0x00000fff          ;程序的大小正好是4KB的倍数吗?
647      cmovnz eax,ebx              ;不是。使用凑整的结果
648
649      mov ecx,eax
650      shr ecx,12                  ;程序占用的总4KB页数
651
652      mov eax,[ebp+40]             ;起始扇区号
653      mov esi,[ebp+36]            ;从堆栈中取得TCB的基地址
654      .b2:
655      alloc_user_linear            ;宏：在用户任务地址空间上分配内存
656
657      push ecx
658      mov ecx,8
659      .b3:
660      call flat_4gb_code_seg_sel:read_hard_disk_0
661      inc eax
662      loop .b3
663
664      pop ecx
665      loop .b2
666
667      ;在内核地址空间内创建用户任务的TSS
668      alloc_core_linear            ;宏：在内核的地址空间上分配内存
669      ;用户任务的TSS必须在全局空间上分配
670
671      mov [esi+0x14],ebx           ;在TCB中填写TSS的线性地址
672      mov word [esi+0x12],103      ;在TCB中填写TSS的界限值
673
674      ;在用户任务的局部地址空间内创建LDT
675      alloc_user_linear            ;宏：在用户任务地址空间上分配内存
676
677      mov [esi+0x0c],ebx           ;填写LDT线性地址到TCB中
678
679      ;建立程序代码段描述符
680      mov eax,0x00000000
681      mov ebx,0x000fffff
682      mov ecx,0x00c0f800          ;4KB粒度的代码段描述符，特权级3
683      call flat_4gb_code_seg_sel:make_seg_descriptor
684      mov ebx,esi                 ;TCB的基地址
685      call fill_descriptor_in_ldt
686      or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
687
688      mov ebx,[esi+0x14]           ;从TCB中获取TSS的线性地址
689      mov [ebx+76],cx             ;填写TSS的CS域
690
691      ;建立程序数据段描述符
692      mov eax,0x00000000
693      mov ebx,0x000fffff
694      mov ecx,0x00c0f200          ;4KB粒度的数据段描述符，特权级3
695      call flat_4gb_code_seg_sel:make_seg_descriptor
696      mov ebx,esi                 ;TCB的基地址
697      call fill_descriptor_in_ldt
698      or cx,0000_0000_0000_0011B ;设置选择子的特权级为3
699
700      mov ebx,[esi+0x14]           ;从TCB中获取TSS的线性地址
701      mov [ebx+84],cx             ;填写TSS的DS域
702      mov [ebx+72],cx             ;填写TSS的ES域
703      mov [ebx+88],cx             ;填写TSS的FS域
704      mov [ebx+92],cx             ;填写TSS的GS域
705
706      ;将数据段作为用户任务的3特权级固有堆栈
707      alloc_user_linear            ;宏：在用户任务地址空间上分配内存
708
709      mov ebx,[esi+0x14]           ;从TCB中获取TSS的线性地址
710      mov [ebx+80],cx             ;填写TSS的SS域
711      mov edx,[esi+0x06]           ;堆栈的高端线性地址
712      mov [ebx+56],edx            ;填写TSS的ESP域
713
714      ;在用户任务的局部地址空间内创建0特权级堆栈
715      alloc_user_linear            ;宏：在用户任务地址空间上分配内存
716
717      mov eax,0x00000000
718      mov ebx,0x000fffff
719      mov ecx,0x00c09200          ;4KB粒度的堆栈段描述符，特权级0
720      call flat_4gb_code_seg_sel:make_seg_descriptor
721      mov ebx,esi                 ;TCB的基地址
722      call fill_descriptor_in_ldt
723      or cx,0000_0000_0000_0000B ;设置选择子的特权级为0
724
725      mov ebx,[esi+0x14]           ;从TCB中获取TSS的线性地址
726      mov [ebx+8],cx              ;填写TSS的SS0域
727      mov edx,[esi+0x06]           ;堆栈的高端线性地址
728      mov [ebx+4],edx             ;填写TSS的ESP0域
729
730      ;在用户任务的局部地址空间内创建1特权级堆栈
731      alloc_user_linear            ;宏：在用户任务地址空间上分配内存
732
733      mov eax,0x00000000
734      mov ebx,0x000fffff
735      mov ecx,0x00c0b200          ;4KB粒度的堆栈段描述符，特权级1
736      call flat_4gb_code_seg_sel:make_seg_descriptor

```

```

737      mov ebx,esi                ;TCB的基地址
738      call fill_descriptor_in_ldt
739      or cx,0000_0000_0000_0001B ;设置选择子的特权级为1
740
741      mov ebx,[esi+0x14]          ;从TCB中获取TSS的线性地址
742      mov [ebx+16],cx             ;填写TSS的SS1域
743      mov edx,[esi+0x06]          ;堆栈的高端线性地址
744      mov [ebx+12],edx            ;填写TSS的ESP1域
745
746      ;在用户任务的局部地址空间内创建2特权级堆栈
747      alloc_user_linear           ;宏：在用户任务地址空间上分配内存
748
749      mov eax,0x00000000
750      mov ebx,0x000fffff
751      mov ecx,0x00c0d200          ;4KB粒度的堆栈段描述符，特权级2
752      call flat_4gb_code_seg_sel:make_seg_descriptor
753      mov ebx,esi                 ;TCB的基地址
754      call fill_descriptor_in_ldt
755      or cx,0000_0000_0000_0010B ;设置选择子的特权级为2
756
757      mov ebx,[esi+0x14]          ;从TCB中获取TSS的线性地址
758      mov [ebx+24],cx             ;填写TSS的SS2域
759      mov edx,[esi+0x06]          ;堆栈的高端线性地址
760      mov [ebx+20],edx            ;填写TSS的ESP2域
761
762      ;重定位U-SALT
763      cld
764
765      mov ecx,[0x0c]              ;U-SALT条目数
766      mov edi,[0x08]              ;U-SALT在4GB空间内的偏移
767      .b4:
768      push ecx
769      push edi
770
771      mov ecx,salt_items
772      mov esi,salt
773      .b5:
774      push edi
775      push esi
776      push ecx
777
778      mov ecx,64                  ;检索表中，每条目的比较次数
779      repe cmpsd                  ;每次比较4字节
780      jnz .b6
781      mov eax,[esi]               ;若匹配，则esi恰好指向其后的地址
782      mov [edi-256],eax           ;将字符串改写成偏移地址
783      mov ax,[esi+4]
784      or ax,0000000000000011B    ;以用户程序自己的特权级使用调用门
785      ;故RPL=3
786      mov [edi-252],ax           ;回填调用门选择子
787      .b6:
788
789      pop ecx
790      pop esi
791      add esi,salt_item_len
792      pop edi                      ;从头比较
793      loop .b5
794
795      pop edi
796      add edi,256
797      pop ecx
798      loop .b4
799
800      ;在GDT中登记LDT描述符
801      mov esi,[ebp+36]            ;从堆栈中取得TCB的基地址
802      mov eax,[esi+0x0c]          ;LDT的起始线性地址
803      movzx ebx,word [esi+0x0a]   ;LDT段界限
804      mov ecx,0x00408200          ;LDT描述符，特权级0
805      call flat_4gb_code_seg_sel:make_seg_descriptor
806      call flat_4gb_code_seg_sel:set_up_gdt_descriptor
807      mov [esi+0x10],cx          ;登记LDT选择子到TCB中
808
809      mov ebx,[esi+0x14]          ;从TCB中获取TSS的线性地址
810      mov [ebx+96],cx             ;填写TSS的LDT域
811
812      mov word [ebx+0],0          ;反向链=0
813
814      mov dx,[esi+0x12]           ;段长度（界限）
815      mov [ebx+102],dx           ;填写TSS的I/0位图偏移域
816
817      mov word [ebx+100],0        ;T=0
818
819      mov eax,[0x04]             ;从任务的4GB地址空间获取入口点
820      mov [ebx+32],eax           ;填写TSS的EIP域
821
822      pushfd
823      pop edx
824      mov [ebx+36],edx           ;填写TSS的EFLAGS域
825
826      ;在GDT中登记TSS描述符
827      mov eax,[esi+0x14]          ;从TCB中获取TSS的起始线性地址
828      movzx ebx,word [esi+0x12]  ;段长度（界限）

```

```

829      mov ecx,0x00408900          ;TSS描述符, 特权级0
830      call flat_4gb_code_seg_sel:make_seg_descriptor
831      call flat_4gb_code_seg_sel:set_up_gdt_descriptor
832      mov [esi+0x18],cx            ;登记TSS选择子到TCB
833
834      ;创建用户任务的项目录
835      ;注意! 页的分配和使用是由页位图决定的, 可以不占用线性地址空间
836      call flat_4gb_code_seg_sel:create_copy_cur_pdir
837      mov ebx,[esi+0x14]           ;从TCB中获取TSS的线性地址
838      mov dword [ebx+28],eax       ;填写TSS的CR3(PDBR)域
839
840      popad
841
842      ret 8                        ;丢弃调用本过程前压入的参数
843
844      ;-----
845      append_to_tcb_link:          ;在TCB链上追加任务控制块
846                                   ;输入: ECX=TCB线性基地址
847      cli
848
849      push eax
850      push ebx
851
852      mov eax,tcb_chain
853      .b0:                         ;EAX=链表头或当前TCB线性地址
854      mov ebx,[eax]               ;EBX=下一个TCB线性地址
855      or ebx,ebx
856      jz .b1                      ;链表为空, 或已到末尾
857      mov eax,ebx                 ;定位到下一个TCB (的线性地址)
858      jmp .b0
859
860      .b1:
861      mov [eax],ecx
862      mov dword [ecx],0x00000000   ;当前TCB指针域清零, 以指示这是最
863                                   ;后一个TCB
864      pop ebx
865      pop eax
866
867      sti
868
869      ret
870
871      ;-----
872      start:
873      ;创建中断描述符表IDT
874      ;在此之前, 禁止调用put_string过程, 以及任何含有sti指令的过程。
875
876      ;前20个向量是处理器异常使用的
877      mov eax,general_exception_handler ;门代码在段内偏移地址
878      mov bx,flat_4gb_code_seg_sel     ;门代码所在段的选择子
879      mov cx,0x8e00                    ;32位中断门, 0特权级
880      call flat_4gb_code_seg_sel:make_gate_descriptor
881
882      mov ebx,idt_linear_address        ;中断描述符表的线性地址
883      xor esi,esi
884      .idt0:
885      mov [ebx+esi*8],eax
886      mov [ebx+esi*8+4],edx
887      inc esi
888      cmp esi,19                       ;安装前20个异常中断处理过程
889      jle .idt0
890
891      ;其余为保留或硬件使用的中断向量
892      mov eax,general_interrupt_handler ;门代码在段内偏移地址
893      mov bx,flat_4gb_code_seg_sel     ;门代码所在段的选择子
894      mov cx,0x8e00                    ;32位中断门, 0特权级
895      call flat_4gb_code_seg_sel:make_gate_descriptor
896
897      mov ebx,idt_linear_address        ;中断描述符表的线性地址
898      .idt1:
899      mov [ebx+esi*8],eax
900      mov [ebx+esi*8+4],edx
901      inc esi
902      cmp esi,255                      ;安装普通的中断处理过程
903      jle .idt1
904
905      ;设置实时时钟中断处理过程
906      mov eax,rtm_0x70_interrupt_handle ;门代码在段内偏移地址
907      mov bx,flat_4gb_code_seg_sel     ;门代码所在段的选择子
908      mov cx,0x8e00                    ;32位中断门, 0特权级
909      call flat_4gb_code_seg_sel:make_gate_descriptor
910
911      mov ebx,idt_linear_address        ;中断描述符表的线性地址
912      mov [ebx+0x70*8],eax
913      mov [ebx+0x70*8+4],edx
914
915      ;准备开放中断
916      mov word [pidt],256*8-1          ;IDT的界限
917      mov dword [pidt+2],idt_linear_address
918      lidt [pidt]                     ;加载中断描述符表寄存器IDTR
919
920      ;设置8259A中断控制器

```

```

921      mov al,0x11
922      out 0x20,al                      ;ICW1: 边沿触发/级联方式
923      mov al,0x20
924      out 0x21,al                      ;ICW2:起始中断向量
925      mov al,0x04
926      out 0x21,al                      ;ICW3:从片级联到IR2
927      mov al,0x01
928      out 0x21,al                      ;ICW4:非总线缓冲, 全嵌套, 正常EOI
929
930      mov al,0x11
931      out 0xa0,al                      ;ICW1: 边沿触发/级联方式
932      mov al,0x70
933      out 0xa1,al                      ;ICW2:起始中断向量
934      mov al,0x04
935      out 0xa1,al                      ;ICW3:从片级联到IR2
936      mov al,0x01
937      out 0xa1,al                      ;ICW4:非总线缓冲, 全嵌套, 正常EOI
938
939      ;设置和时钟中断相关的硬件
940      mov al,0x0b                      ;RTC寄存器B
941      or al,0x80                      ;阻断NMI
942      out 0x70,al
943      mov al,0x12                      ;设置寄存器B, 禁止周期性中断, 开放更
944      out 0x71,al                      ;新结束后中断, BCD码, 24小时制
945
946      in al,0xa1                      ;读8259从片的IMR寄存器
947      and al,0xfe                      ;清除bit 0(此位连接RTC)
948      out 0xa1,al                      ;写回此寄存器
949
950      mov al,0x0c
951      out 0x70,al
952      in al,0x71                      ;读RTC寄存器C, 复位未决的中断状态
953
954      sti                             ;开放硬件中断
955
956      mov ebx,message_0
957      call flat_4gb_code_seg_sel:put_string
958
959      ;显示处理器品牌信息
960      mov eax,0x80000002
961      cpuid
962      mov [cpu_brand + 0x00],eax
963      mov [cpu_brand + 0x04],ebx
964      mov [cpu_brand + 0x08],ecx
965      mov [cpu_brand + 0x0c],edx
966
967      mov eax,0x80000003
968      cpuid
969      mov [cpu_brand + 0x10],eax
970      mov [cpu_brand + 0x14],ebx
971      mov [cpu_brand + 0x18],ecx
972      mov [cpu_brand + 0x1c],edx
973
974      mov eax,0x80000004
975      cpuid
976      mov [cpu_brand + 0x20],eax
977      mov [cpu_brand + 0x24],ebx
978      mov [cpu_brand + 0x28],ecx
979      mov [cpu_brand + 0x2c],edx
980
981      mov ebx,cpu_brnd0                ;显示处理器品牌信息
982      call flat_4gb_code_seg_sel:put_string
983      mov ebx,cpu_brand
984      call flat_4gb_code_seg_sel:put_string
985      mov ebx,cpu_brnd1
986      call flat_4gb_code_seg_sel:put_string
987
988      ;以下开始安装为整个系统服务的调用门。特权级之间的控制转移必须使用门
989      mov edi,salt                     ;C-SALT表的起始位置
990      mov ecx,salt_items               ;C-SALT表的条目数量
991
992      .b4:
993      push ecx
994      mov eax,[edi+256]                ;该条目入口点的32位偏移地址
995      mov bx,[edi+260]                 ;该条目入口点的段选择子
996      mov cx,1_11_0_1100_000_00000B ;特权级3的调用门(3以上的特权级才
997                                      ;允许访问), 0个参数(因为用寄存器
998                                      ;传递参数, 而没有用栈)
999      call flat_4gb_code_seg_sel:make_gate_descriptor
1000     call flat_4gb_code_seg_sel:set_up_gdt_descriptor
1001     mov [edi+260],cx                 ;将返回的门描述符选择子回填
1002     add edi,salt_item_len            ;指向下一个C-SALT条目
1003     pop ecx
1004     loop .b4
1005
1006     ;对门进行测试
1007     mov ebx,message_1
1008     call far [salt_1+256]             ;通过门显示信息(偏移量将被忽略)
1009
1010     ;初始化创建程序管理器任务的任务控制块TCB
1011     mov word [core_tcb+0x04],0xffff   ;任务状态: 忙碌
1012     mov dword [core_tcb+0x06],0x80100000 ;内核虚拟空间的分配从这里开始。

```

```

1013     mov word [core_tcb+0x0a],0xffff    ;登记LDT初始的界限到TCB中（未使用）
1014     mov ecx,core_tcb
1015     call append_to_tcb_link            ;将此TCB添加到TCB链中
1016
1017     ;为程序管理器的TSS分配内存空间
1018     alloc_core_linear                  ;宏：在内核的虚拟地址空间分配内存
1019
1020     ;在程序管理器的TSS中设置必要的项目
1021     mov word [ebx+0],0                 ;反向链=0
1022     mov eax,cr3
1023     mov dword [ebx+28],eax             ;登记CR3(PDBR)
1024     mov word [ebx+96],0                ;没有LDT。处理器允许没有LDT的任务。
1025     mov word [ebx+100],0               ;T=0
1026     mov word [ebx+102],103             ;没有I/O位图。0特权级事实上不需要。
1027
1028     ;创建程序管理器的TSS描述符，并安装到GDT中
1029     mov eax,ebx                        ;TSS的起始线性地址
1030     mov ebx,103                        ;段长度（界限）
1031     mov ecx,0x00408900                ;TSS描述符，特权级0
1032     call flat_4gb_code_seg_sel:make_seg_descriptor
1033     call flat_4gb_code_seg_sel:set_up_gdt_descriptor
1034     mov [core_tcb+0x18],cx             ;登记内核任务的TSS选择子到其TCB
1035
1036     ;任务寄存器TR中的内容是任务存在的标志，该内容也决定了当前任务是谁。
1037     ;下面的指令为当前正在执行的0特权级任务“程序管理器”后补手续（TSS）。
1038     ltr cx
1039
1040     ;现在可认为“程序管理器”任务正执行中
1041
1042     ;创建用户任务的任务控制块
1043     alloc_core_linear                  ;宏：在内核的虚拟地址空间分配内存
1044
1045     mov word [ebx+0x04],0              ;任务状态：空闲
1046     mov dword [ebx+0x06],0             ;用户任务局部空间的分配从0开始。
1047     mov word [ebx+0x0a],0xffff         ;登记LDT初始的界限到TCB中
1048
1049     push dword 50                      ;用户程序位于逻辑50扇区
1050     push ebx                           ;压入任务控制块起始线性地址
1051     call load_relocate_program
1052     mov ecx,ebx
1053     call append_to_tcb_link            ;将此TCB添加到TCB链中
1054
1055     ;创建用户任务的任务控制块
1056     alloc_core_linear                  ;宏：在内核的虚拟地址空间分配内存
1057
1058     mov word [ebx+0x04],0              ;任务状态：空闲
1059     mov dword [ebx+0x06],0             ;用户任务局部空间的分配从0开始。
1060     mov word [ebx+0x0a],0xffff         ;登记LDT初始的界限到TCB中
1061
1062     push dword 100                     ;用户程序位于逻辑100扇区
1063     push ebx                           ;压入任务控制块起始线性地址
1064     call load_relocate_program
1065     mov ecx,ebx
1066     call append_to_tcb_link            ;将此TCB添加到TCB链中
1067
1068     .core:
1069     mov ebx,core_msg0
1070     call flat_4gb_code_seg_sel:put_string
1071
1072     ;这里可以编写回收已终止任务内存的代码
1073
1074     jmp .core
1075
1076     core_code_end:
1077
1078     ;-----
1079     SECTION core_trail
1080     ;-----
1081     core_end:

```

```
1      ;代码清单17-1
2      ;文件名: c17_mbr.asm
3      ;文件说明: 硬盘主引导扇区代码
4      ;创建日期: 2012-07-13 11:20      ;设置堆栈段和栈指针
5
6      core_base_address equ 0x00040000 ;常数, 内核加载的起始内存地址
7      core_start_sector equ 0x00000001 ;常数, 内核的起始逻辑扇区号
8
9      ;=====
10     SECTION mbr vstart=0x0007c00
11
12     mov ax,cs
13     mov ss,ax
14     mov sp,0x7c00
15
16     ;计算GDT所在的逻辑段地址
17     mov eax,[cs:pgdt+0x02]      ;GDT的32位物理地址
18     xor edx,edx
19     mov ebx,16
20     div ebx      ;分解成16位逻辑地址
21
22     mov ds,eax      ;令DS指向该段以进行操作
23     mov ebx,edx      ;段内起始偏移地址
24
25     ;跳过0#号描述符的槽位
26     ;创建1#描述符, 保护模式下的代码段描述符
27     mov dword [ebx+0x08],0x0000ffff ;基地址为0, 界限0xFFFF, DPL=00
28     mov dword [ebx+0x0c],0x00cf9800 ;4KB粒度, 代码段描述符, 向上扩展
29
30     ;创建2#描述符, 保护模式下的数据段和堆栈段描述符
31     mov dword [ebx+0x10],0x0000ffff ;基地址为0, 界限0xFFFF, DPL=00
32     mov dword [ebx+0x14],0x00cf9200 ;4KB粒度, 数据段描述符, 向上扩展
33
34     ;初始化描述符表寄存器GDTR
35     mov word [cs: pgdt],23      ;描述符表的界限
36
37     lgdt [cs: pgdt]
38
39     in al,0x92      ;南桥芯片内的端口
40     or al,0000_0010B
41     out 0x92,al      ;打开A20
42
43     cli      ;中断机制尚未工作
44
45     mov eax,cr0
46     or eax,1
47     mov cr0,eax      ;设置PE位
48
49     ;以下进入保护模式...
50     jmp dword 0x0008:flush      ;16位的描述符选择子: 32位偏移
51                                ;流水线并串行化处理器
52
53     flush:
54     mov eax,0x00010      ;加载数据段(4GB)选择子
55     mov ds,eax
56     mov es,eax
57     mov fs,eax
58     mov gs,eax
59     mov ss,eax      ;加载堆栈段(4GB)选择子
60     mov esp,0x7000      ;堆栈指针
61
62     ;以下加载系统核心程序
63     mov edi,core_base_address
64
65     mov eax,core_start_sector
66     mov ebx,edi      ;起始地址
67     call read_hard_disk_0      ;以下读取程序的起始部分(一个扇区)
68
69     ;以下判断整个程序有多大
70     mov eax,[edi]      ;核心程序尺寸
71     xor edx,edx
72     mov ecx,512      ;512字节每扇区
73     div ecx
74
75     or edx,edx
76     jnz @1      ;未除尽, 因此结果比实际扇区数少1
77     dec eax      ;已经读了一个扇区, 扇区总数减1
78
79     @1:
80     or eax,eax      ;考虑实际长度≤512个字节的情况
81     jz pge      ;EAX=0 ?
82
83     ;读取剩余的扇区
84     mov ecx,eax      ;32位模式下的LOOP使用ECX
85     mov eax,core_start_sector
86     inc eax      ;从下一个逻辑扇区接着读
87
88     @2:
89     call read_hard_disk_0
90     inc eax
91     loop @2      ;循环读, 直到读完整个内核
92
93     pge:
94     ;准备打开分页机制。从此, 再也不用在段之间转来转去, 实在晕乎~
```

```
93
94 ;创建系统内核的页目录表PDT
95 mov ebx,0x00020000 ;页目录表PDT的物理地址
96
97 ;在页目录内创建指向页目录表自己的目录项
98 mov dword [ebx+4092],0x00020003
99
100 mov edx,0x00021003 ;MBR空间有限，后面尽量不使用立即数
101 ;在页目录内创建与线性地址0x00000000对应的目录项
102 mov [ebx+0x000],edx ;写入目录项（页表的物理地址和属性）
103 ;此目录项仅用于过渡。
104 ;在页目录内创建与线性地址0x80000000对应的目录项
105 mov [ebx+0x800],edx ;写入目录项（页表的物理地址和属性）
106
107 ;创建与上面那个目录项相对应的页表，初始化页表项
108 mov ebx,0x00021000 ;页表的物理地址
109 xor eax,eax ;起始页的物理地址
110 xor esi,esi
111 .b1:
112 mov edx,eax
113 or edx,0x00000003
114 mov [ebx+esi*4],edx ;登记页的物理地址
115 add eax,0x1000 ;下一个相邻页的物理地址
116 inc esi
117 cmp esi,256 ;仅低端1MB内存对应的页才是有效的
118 jl .b1
119
120 ;令CR3寄存器指向页目录，并正式开启页功能
121 mov eax,0x00020000 ;PCD=PWT=0
122 mov cr3,eax
123
124 ;将GDT的线性地址映射到从0x80000000开始的相同位置
125 sgdt [pgdt]
126 mov ebx,[pgdt+2]
127 add dword [pgdt+2],0x80000000 ;GDTR也用的是线性地址
128 lgdt [pgdt]
129
130 mov eax,cr0
131 or eax,0x80000000
132 mov cr0,eax ;开启分页机制
133
134 ;将堆栈映射到高端，这是非常容易被忽略的一件事。应当把内核的所有东西
135 ;都移到高端，否则，一定会和正在加载的用户任务局部空间里的内容冲突，
136 ;而且很难想到问题会出在这里。
137 add esp,0x80000000
138
139 jmp [0x80040004]
140
141 ;-----
142 read_hard_disk_0: ;从硬盘读取一个逻辑扇区
143 ;EAX=逻辑扇区号
144 ;DS:EBX=目标缓冲区地址
145 ;返回: EBX=EBX+512
146
147 push eax
148 push ecx
149 push edx
150
151 push eax
152
153 mov dx,0x1f2
154 mov al,1
155 out dx,al ;读取的扇区数
156
157 inc dx ;0x1f3
158 pop eax
159 out dx,al ;LBA地址7~0
160
161 inc dx ;0x1f4
162 mov cl,8
163 shr eax,cl
164 out dx,al ;LBA地址15~8
165
166 inc dx ;0x1f5
167 shr eax,cl
168 out dx,al ;LBA地址23~16
169
170 inc dx ;0x1f6
171 shr eax,cl
172 or al,0xe0 ;第一硬盘 LBA地址27~24
173 out dx,al
174
175 inc dx ;0x1f7
176 mov al,0x20 ;读命令
177 out dx,al
178
179 .waits:
180 in al,dx
181 and al,0x88
182 cmp al,0x08
183 jnz .waits ;不忙，且硬盘已准备好数据传输
184
185 mov ecx,256 ;总共要读取的字数
```



```
185         mov dx,0x1f0
186     .readw:
187         in ax,dx
188         mov [ebx],ax
189         add ebx,2
190         loop .readw
191
192         pop edx
193         pop ecx
194         pop eax
195
196         ret
197
198 ;-----
199         pgdt             dw 0
200                         dd 0x00008000 ;GDT的物理/线性地址
201 ;-----
202         times 510-($-$$) db 0
203                         db 0x55,0xaa
```



```
1      ;代码清单17-3
2      ;文件名: c17 1.asm
3      ;文件说明: 用户程序
4      ;创建日期: 2012-07-14 15:46
5
6      program_length dd program_end      ;程序总长度#0x00
7      entry_point   dd start            ;程序入口点#0x04
8      salt_position  dd salt_begin       ;SALT表起始偏移量#0x08
9      salt_items     dd (salt_end-salt_begin)/256 ;SALT条目数#0x0C
10
11 ;-----
12
13 ;符号地址检索表
14 salt_begin:
15
16     PrintString     db '@PrintString'
17                     times 256-($-PrintString) db 0
18
19     TerminateProgram db '@TerminateProgram'
20                     times 256-($-TerminateProgram) db 0
21
22     ReadDiskData     db '@ReadDiskData'
23                     times 256-($-ReadDiskData) db 0
24
25     PrintDwordAsHex db '@PrintDwordAsHexString'
26                     times 256-($-PrintDwordAsHex) db 0
27
28     salt_end:
29
30     message_0        db ' User task A->;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;'
31                     db 0x0d,0x0a,0
32
33 ;-----
34 [bits 32]
35 ;-----
36
37 start:
38
39     mov ebx,message_0
40     call far [PrintString]
41     jmp start
42
43     call far [TerminateProgram]      ;退出，并将控制权返回到核心
44
45 ;-----
46 program_end:
```



```
1      ;代码清单17-4
2      ;文件名: c17 2.asm
3      ;文件说明: 用户程序
4      ;创建日期: 2012-07-16 12:27
5
6      program_length dd program_end      ;程序总长度#0x00
7      entry_point   dd start            ;程序入口点#0x04
8      salt_position  dd salt_begin       ;SALT表起始偏移量#0x08
9      salt_items     dd (salt_end-salt_begin)/256 ;SALT条目数#0x0C
10
11 ;-----
12
13 ;符号地址检索表
14 salt_begin:
15
16     PrintString     db '@PrintString'
17                     times 256-($-PrintString) db 0
18
19     TerminateProgram db '@TerminateProgram'
20                     times 256-($-TerminateProgram) db 0
21
22     ReadDiskData     db '@ReadDiskData'
23                     times 256-($-ReadDiskData) db 0
24
25     PrintDwordAsHex  db '@PrintDwordAsHexString'
26                     times 256-($-PrintDwordAsHex) db 0
27
28     salt_end:
29
30     message_0        db ' User task B->$$$$$$$$$$$$$$$$$$$$'
31                     db 0x0d,0x0a,0
32
33 ;-----
34 [bits 32]
35 ;-----
36
37 start:
38
39     mov ebx,message_0
40     call far [PrintString]
41     jmp start
42
43     call far [TerminateProgram]      ;退出，并将控制权返回到核心
44
45 ;-----
46 program_end:
```

