```
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
```

```
                                        ;光标。仅用于段内调用
                                        ;输入：CL=字符ASCII码
        pushad

        ;以下取当前光标位置
        mov dx,0x3d4
        mov al,0x0e
        out dx,al
        inc dx                          ;0x3d5
        in al,dx                        ;高字
        mov ah,al

        dec dx                          ;0x3d4
        mov al,0x0f
        out dx,al
        inc dx                          ;0x3d5
        in al,dx                        ;低字
        mov bx,ax                       ;BX=代表光标位置的16位数

        cmp cl,0x0d                     ;回车符?
        jnz .put_0a
        mov ax,bx
        mov bl,80
        div bl
        mul bl
        mov bx,ax
        jmp .set_cursor

.put_0a:
        cmp cl,0x0a                     ;换行符?
        jnz .put_other
        add bx,80
        jmp .roll_screen

.put_other:                             ;正常显示字符
        push es
        mov eax,video_ram_seg_sel       ;0xb8000段的选择子
        mov es,eax
        shl bx,1
        mov [es:bx],cl
        pop es

        ;以下将光标位置推进一个字符
        shr bx,1
        inc bx

.roll_screen:
        cmp bx,2000                     ;光标超出屏幕？滚屏
        jl .set_cursor

        push ds
        push es
        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=目标缓冲区地址
147                                             ;返回：EBX=EBX+512
148         push eax
149         push ecx
150         push edx
151
152         push eax
153
154         mov dx,0x1f2
155         mov al,1
156         out dx,al                           ;读取的扇区数
157
158         inc dx                              ;0x1f3
159         pop eax
```

```
160         out dx,al                           ;LBA地址7~0
161
162         inc dx                              ;0x1f4
163         mov cl,8
164         shr eax,cl
165         out dx,al                           ;LBA地址15~8
166
167         inc dx                              ;0x1f5
168         shr eax,cl
169         out dx,al                           ;LBA地址23~16
170
171         inc dx                              ;0x1f6
172         shr eax,cl
173         or al,0xe0                          ;第一硬盘  LBA地址27~24
174         out dx,al
175
176         inc dx                              ;0x1f7
177         mov al,0x20                         ;读命令
178         out dx,al
179
180   .waits:
181         in al,dx
182         and al,0x88
183         cmp al,0x08
184         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,0xfffffffc
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         mov edx,eax
315         shl eax,16
316         or ax,bx                            ;描述符前32位(EAX)构造完毕
317
318         and edx,0xffff0000                  ;清除基地址中无关的位
```

```
319         rol edx,8
320         bswap edx                       ;装配基址的31~24和23~16  (80486+)
321
322         xor bx,bx
323         or edx,ebx                      ;装配段界限的高4位
324
325         or edx,ecx                      ;装配属性
326
327         retf
328
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         push ebx
391         push ecx
392         push edx
393         push esi
394         push edi
395
396         push ds
397         push es
398
399         mov eax,core_data_seg_sel
400         mov ds,eax                              ;切换DS到内核数据段
401
402         mov eax,esi                             ;读取程序头部数据
403         mov ebx,core_buf
404         call sys_routine_seg_sel:read_hard_disk_0
405
406         ;以下判断整个程序有多大
407         mov eax,[core_buf]                      ;程序尺寸
408         mov ebx,eax
409         and ebx,0xfffffe00                      ;使之512字节对齐（能被512整除的数，
410         add ebx,512                             ;低9位都为0
411         test eax,0x000001ff                     ;程序的大小正好是512的倍数吗?
412         cmovnz eax,ebx                          ;不是。使用凑整的结果
413
414         mov ecx,eax                             ;实际需要申请的内存数量
415         call sys_routine_seg_sel:allocate_memory
416         mov ebx,ecx                             ;ebx -> 申请到的内存首地址
417         push ebx                                ;保存该首地址
418         xor edx,edx
419         mov ecx,512
420         div ecx
421         mov ecx,eax                             ;总扇区数
422
423         mov eax,mem_0_4_gb_seg_sel              ;切换DS到0-4GB的段
424         mov ds,eax
```

```
425
426         mov eax,esi                             ;起始扇区号
427     .b1:
428         call sys_routine_seg_sel:read_hard_disk_0
429         inc eax
430         loop .b1                                ;循环读，直到读完整个用户程序
431
432         ;建立程序头部段描述符
433         pop edi                                 ;恢复程序装载的首地址
434         mov eax,edi                             ;程序头部起始线性地址
435         mov ebx,[edi+0x04]                      ;段长度
436         dec ebx                                 ;段界限
437         mov ecx,0x00409200                      ;字节粒度的数据段描述符
438         call sys_routine_seg_sel:make_seg_descriptor
439         call sys_routine_seg_sel:set_up_gdt_descriptor
440         mov [edi+0x04],cx
441
442         ;建立程序代码段描述符
443         mov eax,edi
444         add eax,[edi+0x14]                      ;代码起始线性地址
445         mov ebx,[edi+0x18]                      ;段长度
446         dec ebx                                 ;段界限
447         mov ecx,0x00409800                      ;字节粒度的代码段描述符
448         call sys_routine_seg_sel:make_seg_descriptor
449         call sys_routine_seg_sel:set_up_gdt_descriptor
450         mov [edi+0x14],cx
451
452         ;建立程序数据段描述符
453         mov eax,edi
454         add eax,[edi+0x1c]                      ;数据段起始线性地址
455         mov ebx,[edi+0x20]                      ;段长度
456         dec ebx                                 ;段界限
457         mov ecx,0x00409200                      ;字节粒度的数据段描述符
458         call sys_routine_seg_sel:make_seg_descriptor
459         call sys_routine_seg_sel:set_up_gdt_descriptor
460         mov [edi+0x1c],cx
461
462         ;建立程序堆栈段描述符
463         mov ecx,[edi+0x0c]                      ;4KB的倍率
464         mov ebx,0x000fffff
465         sub ebx,ecx                             ;得到段界限
466         mov eax,4096
467         mul dword [edi+0x0c]
468         mov ecx,eax                             ;准备为堆栈分配内存
469         call sys_routine_seg_sel:allocate_memory
470         add eax,ecx                             ;得到堆栈的高端物理地址
471         mov ecx,0x00c09600                      ;4KB粒度的堆栈段描述符
472         call sys_routine_seg_sel:make_seg_descriptor
473         call sys_routine_seg_sel:set_up_gdt_descriptor
474         mov [edi+0x08],cx
475
476         ;重定位SALT
477         mov eax,[edi+0x04]
```

```
478         mov es,eax                          ;es -> 用户程序头部
479         mov eax,core_data_seg_sel
480         mov ds,eax
481
482         cld
483
484         mov ecx,[es:0x24]                   ;用户程序的SALT条目数
485         mov edi,0x28                        ;用户程序内的SALT位于头部内0x2c处
486   .b2:
487         push ecx
488         push edi
489
490         mov ecx,salt_items
491         mov esi,salt
492   .b3:
493         push edi
494         push esi
495         push ecx
496
497         mov ecx,64                          ;检索表中，每条目的比较次数
498         repe cmpsd                          ;每次比较4字节
499         jnz .b4
500         mov eax,[esi]                       ;若匹配，esi恰好指向其后的地址数据
501         mov [es:edi-256],eax                ;将字符串改写成偏移地址
502         mov ax,[esi+4]
503         mov [es:edi-252],ax                 ;以及段选择子
504   .b4:
505
506         pop ecx
507         pop esi
508         add esi,salt_item_len
509         pop edi                             ;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:
```