```
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          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   ;0x800b8000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         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
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
298         retf
299
300 ;-------------------------------------------------------------------------------
301 make_gate_descriptor:                           ;构造门的描述符（调用门等）
302                                                 ;输入：EAX=门代码在段内偏移地址
303                                                 ;       BX=门代码所在段的选择子
304                                                 ;       CX=段类型及属性等（各属
305                                                 ;           性位都在原始位置）
306                                                 ;返回：EDX:EAX=完整的描述符
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          push ebx
328          push ecx
329          push edx
330          push ds
331
332          mov eax,core_data_seg_sel
333          mov ds,eax
334
335          xor eax,eax
336    .b1:
337          bts [page_bit_map],eax
338          jnc .b2
339          inc eax
340          cmp eax,page_map_len*8
341          jl .b1
342
343          mov ebx,message_3
344          call sys_routine_seg_sel:put_string
345          hlt                                    ;没有可以分配的页，停机
346
347    .b2:
348          shl eax,12                             ;乘以4096（0x1000）
349
350          pop ds
351          pop edx
352          pop ecx
353          pop ebx
354
355          ret
356
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          push eax
362          push ebx
363          push esi
364          push ds
365
366          mov eax,mem_0_4_gb_seg_sel
367          mov ds,eax
368
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,0xffc00000
```

```
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         push ds
410         push es
411         push esi
412         push edi
413         push ebx
414         push ecx
415
416         mov ebx,mem_0_4_gb_seg_sel
417         mov ds,ebx
418         mov es,ebx
419
420         call allocate_a_4k_page
421         mov ebx,eax
422         or ebx,0x00000007
423         mov [0xfffffff8],ebx
424
```

```
425         mov esi,0xfffff000                  ;ESI->当前页目录的线性地址
426         mov edi,0xfffe000                   ;EDI->新页目录的线性地址
427         mov ecx,1024                        ;ECX=要复制的目录项数
428         cld
429         repe movsd
430
431         pop ecx
432         pop ebx
433         pop edi
434         pop esi
435         pop es
436         pop ds
437
438         retf
439
440 ;-------------------------------------------------------------------------------
441 terminate_current_task:                     ;终止当前任务
442                                             ;注意，执行此例程时，当前任务仍在
443                                             ;运行中。此例程其实也是当前任务的
444                                             ;一部分
445         mov eax,core_data_seg_sel
446         mov ds,eax
447
448         pushfd
449         pop edx
450
451         test dx,0100_0000_0000_0000B        ;测试NT位
452         jnz .b1                             ;当前任务是嵌套的，到.b1执行iretd
453         jmp far [program_man_tss]           ;程序管理器任务
454   .b1:
455         iretd
456
457 sys_routine_end:
458
459 ;===============================================================================
460 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         push eax
543         push edx
544         push edi
545         push ds
546
547         mov ecx,mem_0_4_gb_seg_sel
548         mov ds,ecx
549
550         mov edi,[ebx+0x0c]              ;获得LDT基地址
551
552         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         pushad
586
587         push ds
588         push es
589
590         mov ebp,esp                         ;为访问通过堆栈传递的参数做准备
591
592         mov ecx,mem_0_4_gb_seg_sel
593         mov es,ecx
594
595         ;清空当前页目录的前半部分（对应低2GB的局部地址空间）
596         mov ebx,0xfffff000
597         xor esi,esi
598   .b1:
599         mov dword [es:ebx+esi*4],0x00000000
600         inc esi
601         cmp esi,512
602         jl .b1
603
604         ;以下开始分配内存并加载用户程序
605         mov eax,core_data_seg_sel
606         mov ds,eax                          ;切换DS到内核数据段
607
608         mov eax,[ebp+12*4]                  ;从堆栈中取出用户程序起始扇区号
609         mov ebx,core_buf                    ;读取程序头部数据
610         call sys_routine_seg_sel:read_hard_disk_0
611
612         ;以下判断整个程序有多大
613         mov eax,[core_buf]                  ;程序尺寸
614         mov ebx,eax
615         and ebx,0xfffff000                  ;使之4KB对齐
616         add ebx,0x1000
617         test eax,0x00000fff                 ;程序的大小正好是4KB的倍数吗?
618         cmovnz eax,ebx                      ;不是。使用凑整的结果
619
620         mov ecx,eax
621         shr ecx,12                          ;程序占用的总4KB页数
622
623         mov eax,mem_0_4_gb_seg_sel          ;切换DS到0-4GB的段
624         mov ds,eax
625
626         mov eax,[ebp+12*4]                  ;起始扇区号
627         mov esi,[ebp+11*4]                  ;从堆栈中取得TCB的基地址
628   .b2:
629         mov ebx,[es:esi+0x06]               ;取得可用的线性地址
630         add dword [es:esi+0x06],0x1000
631         call sys_routine_seg_sel:alloc_inst_a_page
632
633         push ecx
634         mov ecx,8
635   .b3:
636         call sys_routine_seg_sel:read_hard_disk_0
```

```
637         inc eax
638         loop .b3
639
640         pop ecx
641         loop .b2
642
643         ;在内核地址空间内创建用户任务的TSS
644         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,0x000fffff
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,0x000fffff
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,0x000fffff
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,0x000fffff
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   .b4:
764         push ecx
765         push edi
766
767         mov ecx,salt_items
768         mov esi,salt
769   .b5:
770         push edi
771         push esi
772         push ecx
773
774         mov ecx,64                          ;检索表中，每条目的比较次数
775         repe cmpsd                          ;每次比较4字节
776         jnz .b6
777         mov eax,[esi]                       ;若匹配，则esi恰好指向其后的地址
778         mov [es:edi-256],eax                ;将字符串改写成偏移地址
779         mov ax,[esi+4]
780         or ax,0000000000000011B             ;以用户程序自己的特权级使用调用门
781                                             ;故RPL=3
782         mov [es:edi-252],ax                 ;回填调用门选择子
783   .b6:
784
785         pop ecx
786         pop esi
787         add esi,salt_item_len
788         pop edi                             ;从头比较
789         loop .b5
790
791         pop edi
792         add edi,256
793         pop ecx
794         loop .b4
795
```

```
796         ;在GDT中登记LDT描述符
797         mov esi,[ebp+11*4]                  ;从堆栈中取得TCB的基地址
798         mov eax,[es:esi+0x0c]               ;LDT的起始线性地址
799         movzx ebx,word [es:esi+0x0a]        ;LDT段界限
800         mov ecx,0x00408200                  ;LDT描述符，特权级0
801         call sys_routine_seg_sel:make_seg_descriptor
802         call sys_routine_seg_sel:set_up_gdt_descriptor
803         mov [es:esi+0x10],cx                ;登记LDT选择子到TCB中
804
805         mov ebx,[es:esi+0x14]               ;从TCB中获取TSS的线性地址
806         mov [es:ebx+96],cx                  ;填写TSS的LDT域
807
808         mov word [es:ebx+0],0               ;反向链=0
809
810         mov dx,[es:esi+0x12]                ;段长度（界限）
811         mov [es:ebx+102],dx                 ;填写TSS的I/O位图偏移域
812
813         mov word [es:ebx+100],0             ;T=0
814
815         mov eax,[es:0x04]                   ;从任务的4GB地址空间获取入口点
816         mov [es:ebx+32],eax                 ;填写TSS的EIP域
817
818         pushfd
819         pop edx
820         mov [es:ebx+36],edx                 ;填写TSS的EFLAGS域
821
822         ;在GDT中登记TSS描述符
823         mov eax,[es:esi+0x14]               ;从TCB中获取TSS的起始线性地址
824         movzx ebx,word [es:esi+0x12]        ;段长度（界限）
825         mov ecx,0x00408900                  ;TSS描述符，特权级0
826         call sys_routine_seg_sel:make_seg_descriptor
827         call sys_routine_seg_sel:set_up_gdt_descriptor
828         mov [es:esi+0x18],cx                ;登记TSS选择子到TCB
829
830         ;创建用户任务的页目录
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   .b1:
931         mov dword [es:ebx+esi],0x00000000   ;页目录表项清零
932         add esi,4
933         loop .b1
934
935         ;在页目录内创建指向页目录自己的目录项
936         mov dword [es:ebx+4092],0x00020003
937
938         ;在页目录内创建与线性地址0x00000000对应的目录项
939         mov dword [es:ebx+0],0x00021003     ;写入目录项（页表的物理地址和属性）
940
941         ;创建与上面那个目录项相对应的页表，初始化页表项
942         mov ebx,0x00021000                  ;页表的物理地址
943         xor eax,eax                         ;起始页的物理地址
944         xor esi,esi
945   .b2:
946         mov edx,eax
947         or edx,0x00000003
948         mov [es:ebx+esi*4],edx              ;登记页的物理地址
949         add eax,0x1000                      ;下一个相邻页的物理地址
950         inc esi
951         cmp esi,256                         ;仅低端1MB内存对应的页才是有效的
952         jl .b2
953
954   .b3:                                      ;其余的页表项置为无效
```

```
955         mov dword [es:ebx+esi*4],0x00000000
956         inc esi
957         cmp esi,1024
958         jl .b3
959
960         ;令CR3寄存器指向页目录，并正式开启页功能
961         mov eax,0x00020000                  ;PCD=PWT=0
962         mov cr3,eax
963
964         mov eax,cr0
965         or eax,0x80000000
966         mov cr0,eax                         ;开启分页机制
967
968         ;在页目录内创建与线性地址0x80000000对应的目录项
969         mov ebx,0xfffff000                  ;页目录自己的线性地址
970         mov esi,0x80000000                  ;映射的起始地址
971         shr esi,22                          ;线性地址的高10位是目录索引
972         shl esi,2
973         mov dword [es:ebx+esi],0x00021003   ;写入目录项（页表的物理地址和属性）
974                                             ;目标单元的线性地址为0xFFFFF200
975
976         ;将GDT中的段描述符映射到线性地址0x80000000
977         sgdt [pgdt]
978
979         mov ebx,[pgdt+2]
980
981         or dword [es:ebx+0x10+4],0x80000000
982         or dword [es:ebx+0x18+4],0x80000000
983         or dword [es:ebx+0x20+4],0x80000000
984         or dword [es:ebx+0x28+4],0x80000000
985         or dword [es:ebx+0x30+4],0x80000000
986         or dword [es:ebx+0x38+4],0x80000000
987
988         add dword [pgdt+2],0x80000000       ;GDTR也用的是线性地址
989
990         lgdt [pgdt]
991
992         jmp core_code_seg_sel:flush         ;刷新段寄存器CS，启用高端线性地址
993
994    flush:
995         mov eax,core_stack_seg_sel
996         mov ss,eax
997
998         mov eax,core_data_seg_sel
999         mov ds,eax
1000
1001        mov ebx,message_1
1002        call sys_routine_seg_sel:put_string
1003
1004        ;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
1005        mov edi,salt                        ;C-SALT表的起始位置
1006        mov ecx,salt_items                  ;C-SALT表的条目数量
1007  .b4:
```

```
1008        push ecx
1009        mov eax,[edi+256]                   ;该条目入口点的32位偏移地址
1010        mov bx,[edi+260]                    ;该条目入口点的段选择子
1011        mov cx,1_11_0_1100_000_00000B       ;特权级3的调用门(3以上的特权级才
1012                                            ;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:
```