```
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
```

```
                                        ;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
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          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,0xfffffffc
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 ;----------------------------------------------------------------------------
331 make_gate_descriptor:                       ;构造门的描述符（调用门等）
332                                            ;输入：EAX=门代码在段内偏移地址
333                                            ;       BX=门代码所在段的选择子
334                                            ;       CX=段类型及属性等（各属
335                                            ;          性位都在原始位置）
336                                            ;返回：EDX:EAX=完整的描述符
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 ;----------------------------------------------------------------------------
354 terminate_current_task:                     ;终止当前任务
355                                            ;注意，执行此例程时，当前任务仍在
356                                            ;运行中。此例程其实也是当前任务的
357                                            ;一部分
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         push eax
470         push edx
471         push edi
472         push ds
473
474         mov ecx,mem_0_4_gb_seg_sel
475         mov ds,ecx
476
477         mov edi,[ebx+0x0c]                  ;获得LDT基地址
```

```
478
479         xor ecx,ecx
480         mov cx,[ebx+0x0a]                   ;获得LDT界限
481         inc cx                              ;LDT的总字节数，即新描述符偏移地址
482
483         mov [edi+ecx+0x00],eax
484         mov [edi+ecx+0x04],edx              ;安装描述符
485
486         add cx,8
487         dec cx                              ;得到新的LDT界限值
488
489         mov [ebx+0x0a],cx                   ;更新LDT界限值到TCB
490
491         mov ax,cx
492         xor dx,dx
493         mov cx,8
494         div cx
495
496         mov cx,ax
497         shl cx,3                            ;左移3位，并且
498         or cx,0000_0000_0000_0100B          ;使TI位=1，指向LDT，最后使RPL=00
499
500         pop ds
501         pop edi
502         pop edx
503         pop eax
504
505         ret
506
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         pushad
513
514         push ds
515         push es
516
517         mov ebp,esp                         ;为访问通过堆栈传递的参数做准备
518
519         mov ecx,mem_0_4_gb_seg_sel
520         mov es,ecx
521
522         mov esi,[ebp+11*4]                  ;从堆栈中取得TCB的基地址
523
524         ;以下申请创建LDT所需要的内存
525         mov ecx,160                         ;允许安装20个LDT描述符
526         call sys_routine_seg_sel:allocate_memory
527         mov [es:esi+0x0c],ecx               ;登记LDT基地址到TCB中
528         mov word [es:esi+0x0a],0xffff       ;登记LDT初始的界限到TCB中
529
530         ;以下开始加载用户程序
```

```
531         mov eax,core_data_seg_sel
532         mov ds,eax                          ;切换DS到内核数据段
533
534         mov eax,[ebp+12*4]                  ;从堆栈中取出用户程序起始扇区号
535         mov ebx,core_buf                    ;读取程序头部数据
536         call sys_routine_seg_sel:read_hard_disk_0
537
538         ;以下判断整个程序有多大
539         mov eax,[core_buf]                  ;程序尺寸
540         mov ebx,eax
541         and ebx,0xfffffe00                  ;使之512字节对齐（能被512整除的数低
542         add ebx,512                         ;9位都为0
543         test eax,0x000001ff                 ;程序的大小正好是512的倍数吗?
544         cmovnz eax,ebx                      ;不是。使用凑整的结果
545
546         mov ecx,eax                         ;实际需要申请的内存数量
547         call sys_routine_seg_sel:allocate_memory
548         mov [es:esi+0x06],ecx               ;登记程序加载基地址到TCB中
549
550         mov ebx,ecx                         ;ebx -> 申请到的内存首地址
551         xor edx,edx
552         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,0xffffe                     ;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,0xffffe                     ;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,0xffffe                     ;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,eax
817         mov eax,mem_0_4_gb_seg_sel          ;令ES指向0..4GB段
818         mov es,eax
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,eax                          ;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   .b3:
891         push ecx
892         mov eax,[edi+256]                   ;该条目入口点的32位偏移地址
893         mov bx,[edi+260]                    ;该条目入口点的段选择子
894         mov cx,1_11_0_1100_000_00000B       ;特权级3的调用门(3以上的特权级才
895                                             ;允许访问)，0个参数(因为用寄存器
896                                             ;传递参数，而没有用栈)
897         call sys_routine_seg_sel:make_gate_descriptor
898         call sys_routine_seg_sel:set_up_gdt_descriptor
899         mov [edi+260],cx                    ;将返回的门描述符选择子回填
900         add edi,salt_item_len               ;指向下一个C-SALT条目
901         pop ecx
```

```
902         loop .b3
903
904         ;对门进行测试
905         mov ebx,message_2
906         call far [salt_1+256]                   ;通过门显示信息(偏移量将被忽略)
907
908         ;为程序管理器的TSS分配内存空间
909         mov ecx,104                             ;为该任务的TSS分配内存
910         call sys_routine_seg_sel:allocate_memory
911         mov [prgman_tss+0x00],ecx               ;保存程序管理器的TSS基地址
912
913         ;在程序管理器的TSS中设置必要的项目
914         mov word [es:ecx+96],0                  ;没有LDT。处理器允许没有LDT的任务。
915         mov word [es:ecx+102],103               ;没有I/O位图。0特权级事实上不需要。
916         mov word [es:ecx+0],0                   ;反向链=0
917         mov dword [es:ecx+28],0                 ;登记CR3(PDBR)
918         mov word [es:ecx+100],0                 ;T=0
919                                                 ;不需要0、1、2特权级堆栈。0特级不
920                                                 ;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:
```