```
1           ;代码清单14-1
2           ;文件名：c14_core.asm
3           ;文件说明：保护模式微型核心程序
4           ;创建日期：2011-11-6 18:37
5
6           ;以下常量定义部分。内核的大部分内容都应当固定
7           core_code_seg_sel     equ  0x38    ;内核代码段选择子
8           core_data_seg_sel     equ  0x30    ;内核数据段选择子
9           sys_routine_seg_sel   equ  0x28    ;系统公共例程代码段的选择子
10          video_ram_seg_sel     equ  0x20    ;视频显示缓冲区的段选择子
11          core_stack_seg_sel    equ  0x18    ;内核堆栈段选择子
12          mem_0_4_gb_seg_sel    equ  0x08    ;整个0-4GB内存的段的选择子
13
14 ;-------------------------------------------------------------------------------
15          ;以下是系统核心的头部，用于加载核心程序
16          core_length      dd core_end       ;核心程序总长度#00
17
18          sys_routine_seg  dd section.sys_routine.start
19                                             ;系统公用例程段位置#04
20
21          core_data_seg    dd section.core_data.start
22                                             ;核心数据段位置#08
23
24          core_code_seg    dd section.core_code.start
25                                             ;核心代码段位置#0c
26
27
28          core_entry       dd start          ;核心代码段入口点#10
29                           dw core_code_seg_sel
30
31 ;===============================================================================
32          [bits 32]
33 ;===============================================================================
34 SECTION sys_routine vstart=0                 ;系统公共例程代码段
35 ;-------------------------------------------------------------------------------
36          ;字符串显示例程
37 put_string:                                  ;显示0终止的字符串并移动光标
38                                             ;输入：DS:EBX=串地址
39          push ecx
40   .getc:
41          mov cl,[ebx]
42          or cl,cl
43          jz .exit
44          call put_char
45          inc ebx
46          jmp .getc
47
48   .exit:
49          pop ecx
50          retf                               ;段间返回
51
52 ;-------------------------------------------------------------------------------
53 put_char:                                   ;在当前光标处显示一个字符,并推进
```

```
                                        ;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 sys_routine_end:
354
355 ;============================================================================
356 SECTION core_data vstart=0                  ;系统核心的数据段
357 ;----------------------------------------------------------------------------
358          pgdt             dw  0             ;用于设置和修改GDT
359                           dd  0
360
361          ram_alloc        dd  0x00100000    ;下次分配内存时的起始地址
362
363          ;符号地址检索表
364          salt:
365          salt_1           db  '@PrintString'
366                      times 256-($-salt_1) db 0
367                           dd  put_string
368                           dw  sys_routine_seg_sel
369
370          salt_2           db  '@ReadDiskData'
371                      times 256-($-salt_2) db 0
```

```
372                             dd  read_hard_disk_0
373                             dw  sys_routine_seg_sel
374
375         salt_3              db  '@PrintDwordAsHexString'
376                         times 256-($-salt_3) db 0
377                             dd  put_hex_dword
378                             dw  sys_routine_seg_sel
379
380         salt_4              db  '@TerminateProgram'
381                         times 256-($-salt_4) db 0
382                             dd  return_point
383                             dw  core_code_seg_sel
384
385         salt_item_len   equ $-salt_4
386         salt_items      equ ($-salt)/salt_item_len
387
388         message_1           db  '  If you seen this message,that means we '
389                             db  'are now in protect mode,and the system '
390                             db  'core is loaded,and the video display '
391                             db  'routine works perfectly.',0x0d,0x0a,0
392
393         message_2           db  '  System wide CALL-GATE mounted.',0x0d,0x0a,0
394
395         message_3           db  0x0d,0x0a,'  Loading user program...',0
396
397         do_status           db  'Done.',0x0d,0x0a,0
398
399         message_6           db  0x0d,0x0a,0x0d,0x0a,0x0d,0x0a
400                             db  '  User program terminated,control returned.',0
401
402         bin_hex             db '0123456789ABCDEF'
403                                                 ;put_hex_dword子过程用的查找表
404
405         core_buf   times 2048 db 0              ;内核用的缓冲区
406
407         esp_pointer         dd 0                ;内核用来临时保存自己的栈指针
408
409         cpu_brnd0           db 0x0d,0x0a,'  ',0
410         cpu_brand  times 52 db 0
411         cpu_brnd1           db 0x0d,0x0a,0x0d,0x0a,0
412
413         ;任务控制块链
414         tcb_chain           dd  0
415
416 core_data_end:
417
418 ;===============================================================================
419 SECTION core_code vstart=0
420 ;-------------------------------------------------------------------------------
421 fill_descriptor_in_ldt:                         ;在LDT内安装一个新的描述符
422                                                 ;输入：EDX:EAX=描述符
423                                                 ;           EBX=TCB基地址
424                                                 ;输出：CX=描述符的选择子
```

```
425         push eax
426         push edx
427         push edi
428         push ds
429
430         mov ecx,mem_0_4_gb_seg_sel
431         mov ds,ecx
432
433         mov edi,[ebx+0x0c]                  ;获得LDT基地址
434
435         xor ecx,ecx
436         mov cx,[ebx+0x0a]                   ;获得LDT界限
437         inc cx                              ;LDT的总字节数，即新描述符偏移地址
438
439         mov [edi+ecx+0x00],eax
440         mov [edi+ecx+0x04],edx              ;安装描述符
441
442         add cx,8
443         dec cx                              ;得到新的LDT界限值
444
445         mov [ebx+0x0a],cx                   ;更新LDT界限值到TCB
446
447         mov ax,cx
448         xor dx,dx
449         mov cx,8
450         div cx
451
452         mov cx,ax
453         shl cx,3                            ;左移3位，并且
454         or cx,0000_0000_0000_0100B          ;使TI位=1，指向LDT，最后使RPL=00
455
456         pop ds
457         pop edi
458         pop edx
459         pop eax
460
461         ret
462
463 ;-------------------------------------------------------------------------------
464 load_relocate_program:                      ;加载并重定位用户程序
465                                             ;输入: PUSH 逻辑扇区号
466                                             ;       PUSH 任务控制块基地址
467                                             ;输出：无
468         pushad
469
470         push ds
471         push es
472
473         mov ebp,esp                         ;为访问通过堆栈传递的参数做准备
474
475         mov ecx,mem_0_4_gb_seg_sel
476         mov es,ecx
477
```

```
478         mov esi,[ebp+11*4]                  ;从堆栈中取得TCB的基地址
479
480         ;以下申请创建LDT所需要的内存
481         mov ecx,160                         ;允许安装20个LDT描述符
482         call sys_routine_seg_sel:allocate_memory
483         mov [es:esi+0x0c],ecx               ;登记LDT基地址到TCB中
484         mov word [es:esi+0x0a],0xffff       ;登记LDT初始的界限到TCB中
485
486         ;以下开始加载用户程序
487         mov eax,core_data_seg_sel
488         mov ds,eax                          ;切换DS到内核数据段
489
490         mov eax,[ebp+12*4]                  ;从堆栈中取出用户程序起始扇区号
491         mov ebx,core_buf                    ;读取程序头部数据
492         call sys_routine_seg_sel:read_hard_disk_0
493
494         ;以下判断整个程序有多大
495         mov eax,[core_buf]                  ;程序尺寸
496         mov ebx,eax
497         and ebx,0xfffffe00                  ;使之512字节对齐（能被512整除的数低
498         add ebx,512                         ;9位都为0
499         test eax,0x000001ff                 ;程序的大小正好是512的倍数吗?
500         cmovnz eax,ebx                      ;不是。使用凑整的结果
501
502         mov ecx,eax                         ;实际需要申请的内存数量
503         call sys_routine_seg_sel:allocate_memory
504         mov [es:esi+0x06],ecx               ;登记程序加载基地址到TCB中
505
506         mov ebx,ecx                         ;ebx -> 申请到的内存首地址
507         xor edx,edx
508         mov ecx,512
509         div ecx
510         mov ecx,eax                         ;总扇区数
511
512         mov eax,mem_0_4_gb_seg_sel          ;切换DS到0-4GB的段
513         mov ds,eax
514
515         mov eax,[ebp+12*4]                  ;起始扇区号
516   .b1:
517         call sys_routine_seg_sel:read_hard_disk_0
518         inc eax
519         loop .b1                            ;循环读，直到读完整个用户程序
520
521         mov edi,[es:esi+0x06]               ;获得程序加载基地址
522
523         ;建立程序头部段描述符
524         mov eax,edi                         ;程序头部起始线性地址
525         mov ebx,[edi+0x04]                  ;段长度
526         dec ebx                             ;段界限
527         mov ecx,0x0040f200                  ;字节粒度的数据段描述符，特权级3
528         call sys_routine_seg_sel:make_seg_descriptor
529
530         ;安装头部段描述符到LDT中
```

```
531         mov ebx,esi                         ;TCB的基地址
532         call fill_descriptor_in_ldt
533
534         or cx,0000_0000_0000_0011B          ;设置选择子的特权级为3
535         mov [es:esi+0x44],cx                ;登记程序头部段选择子到TCB
536         mov [edi+0x04],cx                   ;和头部内
537
538         ;建立程序代码段描述符
539         mov eax,edi
540         add eax,[edi+0x14]                  ;代码起始线性地址
541         mov ebx,[edi+0x18]                  ;段长度
542         dec ebx                             ;段界限
543         mov ecx,0x0040f800                  ;字节粒度的代码段描述符，特权级3
544         call sys_routine_seg_sel:make_seg_descriptor
545         mov ebx,esi                         ;TCB的基地址
546         call fill_descriptor_in_ldt
547         or cx,0000_0000_0000_0011B          ;设置选择子的特权级为3
548         mov [edi+0x14],cx                   ;登记代码段选择子到头部
549
550         ;建立程序数据段描述符
551         mov eax,edi
552         add eax,[edi+0x1c]                  ;数据段起始线性地址
553         mov ebx,[edi+0x20]                  ;段长度
554         dec ebx                             ;段界限
555         mov ecx,0x0040f200                  ;字节粒度的数据段描述符，特权级3
556         call sys_routine_seg_sel:make_seg_descriptor
557         mov ebx,esi                         ;TCB的基地址
558         call fill_descriptor_in_ldt
559         or cx,0000_0000_0000_0011B          ;设置选择子的特权级为3
560         mov [edi+0x1c],cx                   ;登记数据段选择子到头部
561
562         ;建立程序堆栈段描述符
563         mov ecx,[edi+0x0c]                  ;4KB的倍率
564         mov ebx,0x000fffff
565         sub ebx,ecx                         ;得到段界限
566         mov eax,4096
567         mul ecx
568         mov ecx,eax                         ;准备为堆栈分配内存
569         call sys_routine_seg_sel:allocate_memory
570         add eax,ecx                         ;得到堆栈的高端物理地址
571         mov ecx,0x00c0f600                  ;字节粒度的堆栈段描述符，特权级3
572         call sys_routine_seg_sel:make_seg_descriptor
573         mov ebx,esi                         ;TCB的基地址
574         call fill_descriptor_in_ldt
575         or cx,0000_0000_0000_0011B          ;设置选择子的特权级为3
576         mov [edi+0x08],cx                   ;登记堆栈段选择子到头部
577
578         ;重定位SALT
579         mov eax,mem_0_4_gb_seg_sel          ;这里和前一章不同，头部段描述符
580         mov es,eax                          ;已安装，但还没有生效，故只能通
581                                             ;过4GB段访问用户程序头部
582         mov eax,core_data_seg_sel
583         mov ds,eax
```

```
584
585         cld
586
587         mov ecx,[es:edi+0x24]                ;U-SALT条目数(通过访问4GB段取得)
588         add edi,0x28                         ;U-SALT在4GB段内的偏移
589   .b2:
590         push ecx
591         push edi
592
593         mov ecx,salt_items
594         mov esi,salt
595   .b3:
596         push edi
597         push esi
598         push ecx
599
600         mov ecx,64                           ;检索表中，每条目的比较次数
601         repe cmpsd                           ;每次比较4字节
602         jnz .b4
603         mov eax,[esi]                        ;若匹配，则esi恰好指向其后的地址
604         mov [es:edi-256],eax                 ;将字符串改写成偏移地址
605         mov ax,[esi+4]
606         or ax,0000000000000011B              ;以用户程序自己的特权级使用调用门
607                                              ;故RPL=3
608         mov [es:edi-252],ax                  ;回填调用门选择子
609   .b4:
610
611         pop ecx
612         pop esi
613         add esi,salt_item_len
614         pop edi                              ;从头比较
615         loop .b3
616
617         pop edi
618         add edi,256
619         pop ecx
620         loop .b2
621
622         mov esi,[ebp+11*4]                   ;从堆栈中取得TCB的基地址
623
624         ;创建0特权级堆栈
625         mov ecx,4096
626         mov eax,ecx                          ;为生成堆栈高端地址做准备
627         mov [es:esi+0x1a],ecx
628         shr dword [es:esi+0x1a],12           ;登记0特权级堆栈尺寸到TCB
629         call sys_routine_seg_sel:allocate_memory
630         add eax,ecx                          ;堆栈必须使用高端地址为基地址
631         mov [es:esi+0x1e],eax                ;登记0特权级堆栈基地址到TCB
632         mov ebx,0xffffe                      ;段长度（界限）
633         mov ecx,0x00c09600                   ;4KB粒度，读写，特权级0
634         call sys_routine_seg_sel:make_seg_descriptor
635         mov ebx,esi                          ;TCB的基地址
636         call fill_descriptor_in_ldt
```

```
637         ;or cx,0000_0000_0000_0000          ;设置选择子的特权级为0
638         mov [es:esi+0x22],cx                ;登记0特权级堆栈选择子到TCB
639         mov dword [es:esi+0x24],0           ;登记0特权级堆栈初始ESP到TCB
640
641         ;创建1特权级堆栈
642         mov ecx,4096
643         mov eax,ecx                         ;为生成堆栈高端地址做准备
644         mov [es:esi+0x28],ecx
645         shr [es:esi+0x28],12                ;登记1特权级堆栈尺寸到TCB
646         call sys_routine_seg_sel:allocate_memory
647         add eax,ecx                         ;堆栈必须使用高端地址为基地址
648         mov [es:esi+0x2c],eax               ;登记1特权级堆栈基地址到TCB
649         mov ebx,0xffffe                     ;段长度（界限）
650         mov ecx,0x00c0b600                  ;4KB粒度，读写，特权级1
651         call sys_routine_seg_sel:make_seg_descriptor
652         mov ebx,esi                         ;TCB的基地址
653         call fill_descriptor_in_ldt
654         or cx,0000_0000_0000_0001           ;设置选择子的特权级为1
655         mov [es:esi+0x30],cx                ;登记1特权级堆栈选择子到TCB
656         mov dword [es:esi+0x32],0           ;登记1特权级堆栈初始ESP到TCB
657
658         ;创建2特权级堆栈
659         mov ecx,4096
660         mov eax,ecx                         ;为生成堆栈高端地址做准备
661         mov [es:esi+0x36],ecx
662         shr [es:esi+0x36],12                ;登记2特权级堆栈尺寸到TCB
663         call sys_routine_seg_sel:allocate_memory
664         add eax,ecx                         ;堆栈必须使用高端地址为基地址
665         mov [es:esi+0x3a],ecx               ;登记2特权级堆栈基地址到TCB
666         mov ebx,0xffffe                     ;段长度（界限）
667         mov ecx,0x00c0d600                  ;4KB粒度，读写，特权级2
668         call sys_routine_seg_sel:make_seg_descriptor
669         mov ebx,esi                         ;TCB的基地址
670         call fill_descriptor_in_ldt
671         or cx,0000_0000_0000_0010           ;设置选择子的特权级为2
672         mov [es:esi+0x3e],cx                ;登记2特权级堆栈选择子到TCB
673         mov dword [es:esi+0x40],0           ;登记2特权级堆栈初始ESP到TCB
674
675         ;在GDT中登记LDT描述符
676         mov eax,[es:esi+0x0c]               ;LDT的起始线性地址
677         movzx ebx,word [es:esi+0x0a]        ;LDT段界限
678         mov ecx,0x00408200                  ;LDT描述符，特权级0
679         call sys_routine_seg_sel:make_seg_descriptor
680         call sys_routine_seg_sel:set_up_gdt_descriptor
681         mov [es:esi+0x10],cx                ;登记LDT选择子到TCB中
682
683         ;创建用户程序的TSS
684         mov ecx,104                         ;tss的基本尺寸
685         mov [es:esi+0x12],cx
686         dec word [es:esi+0x12]              ;登记TSS界限值到TCB
687         call sys_routine_seg_sel:allocate_memory
688         mov [es:esi+0x14],ecx               ;登记TSS基地址到TCB
689
```

```
690         ;登记基本的TSS表格内容
691         mov word [es:ecx+0],0                   ;反向链=0
692
693         mov edx,[es:esi+0x24]                   ;登记0特权级堆栈初始ESP
694         mov [es:ecx+4],edx                      ;到TSS中
695
696         mov dx,[es:esi+0x22]                    ;登记0特权级堆栈段选择子
697         mov [es:ecx+8],dx                       ;到TSS中
698
699         mov edx,[es:esi+0x32]                   ;登记1特权级堆栈初始ESP
700         mov [es:ecx+12],edx                     ;到TSS中
701
702         mov dx,[es:esi+0x30]                    ;登记1特权级堆栈段选择子
703         mov [es:ecx+16],dx                      ;到TSS中
704
705         mov edx,[es:esi+0x40]                   ;登记2特权级堆栈初始ESP
706         mov [es:ecx+20],edx                     ;到TSS中
707
708         mov dx,[es:esi+0x3e]                    ;登记2特权级堆栈段选择子
709         mov [es:ecx+24],dx                      ;到TSS中
710
711         mov dx,[es:esi+0x10]                    ;登记任务的LDT选择子
712         mov [es:ecx+96],dx                      ;到TSS中
713
714         mov dx,[es:esi+0x12]                    ;登记任务的I/O位图偏移
715         mov [es:ecx+102],dx                     ;到TSS中
716
717         mov word [es:ecx+100],0                 ;T=0
718
719         ;在GDT中登记TSS描述符
720         mov eax,[es:esi+0x14]                   ;TSS的起始线性地址
721         movzx ebx,word [es:esi+0x12]            ;段长度（界限）
722         mov ecx,0x00408900                      ;TSS描述符，特权级0
723         call sys_routine_seg_sel:make_seg_descriptor
724         call sys_routine_seg_sel:set_up_gdt_descriptor
725         mov [es:esi+0x18],cx                    ;登记TSS选择子到TCB
726
727         pop es                                  ;恢复到调用此过程前的es段
728         pop ds                                  ;恢复到调用此过程前的ds段
729
730         popad
731
732         ret 8                                   ;丢弃调用本过程前压入的参数
733
734 ;-------------------------------------------------------------------------------
735 append_to_tcb_link:                             ;在TCB链上追加任务控制块
736                                                 ;输入：ECX=TCB线性基地址
737         push eax
738         push edx
739         push ds
740         push es
741
742         mov eax,core_data_seg_sel               ;令DS指向内核数据段
```

```
743         mov ds,eax
744         mov eax,mem_0_4_gb_seg_sel          ;令ES指向0..4GB段
745         mov es,eax
746
747         mov dword [es: ecx+0x00],0          ;当前TCB指针域清零，以指示这是最
748                                             ;后一个TCB
749
750         mov eax,[tcb_chain]                 ;TCB表头指针
751         or eax,eax                          ;链表为空？
752         jz .notcb
753
754   .searc:
755         mov edx,eax
756         mov eax,[es: edx+0x00]
757         or eax,eax
758         jnz .searc
759
760         mov [es: edx+0x00],ecx
761         jmp .retpc
762
763   .notcb:
764         mov [tcb_chain],ecx                 ;若为空表，直接令表头指针指向TCB
765
766   .retpc:
767         pop es
768         pop ds
769         pop edx
770         pop eax
771
772         ret
773
774 ;--------------------------------------------------------------------------------
775 start:
776         mov ecx,core_data_seg_sel           ;使ds指向核心数据段
777         mov ds,ecx
778
779         mov ebx,message_1
780         call sys_routine_seg_sel:put_string
781
782         ;显示处理器品牌信息
783         mov eax,0x80000002
784         cpuid
785         mov [cpu_brand + 0x00],eax
786         mov [cpu_brand + 0x04],ebx
787         mov [cpu_brand + 0x08],ecx
788         mov [cpu_brand + 0x0c],edx
789
790         mov eax,0x80000003
791         cpuid
792         mov [cpu_brand + 0x10],eax
793         mov [cpu_brand + 0x14],ebx
794         mov [cpu_brand + 0x18],ecx
795         mov [cpu_brand + 0x1c],edx
```

```
796
797         mov eax,0x80000004
798         cpuid
799         mov [cpu_brand + 0x20],eax
800         mov [cpu_brand + 0x24],ebx
801         mov [cpu_brand + 0x28],ecx
802         mov [cpu_brand + 0x2c],edx
803
804         mov ebx,cpu_brnd0                       ;显示处理器品牌信息
805         call sys_routine_seg_sel:put_string
806         mov ebx,cpu_brand
807         call sys_routine_seg_sel:put_string
808         mov ebx,cpu_brnd1
809         call sys_routine_seg_sel:put_string
810
811         ;以下开始安装为整个系统服务的调用门。特权级之间的控制转移必须使用门
812         mov edi,salt                            ;C-SALT表的起始位置
813         mov ecx,salt_items                      ;C-SALT表的条目数量
814   .b3:
815         push ecx
816         mov eax,[edi+256]                       ;该条目入口点的32位偏移地址
817         mov bx,[edi+260]                        ;该条目入口点的段选择子
818         mov cx,1_11_0_1100_000_00000B           ;特权级3的调用门(3以上的特权级才
819                                                 ;允许访问)，0个参数(因为用寄存器
820                                                 ;传递参数，而没有用栈)
821         call sys_routine_seg_sel:make_gate_descriptor
822         call sys_routine_seg_sel:set_up_gdt_descriptor
823         mov [edi+260],cx                        ;将返回的门描述符选择子回填
824         add edi,salt_item_len                   ;指向下一个C-SALT条目
825         pop ecx
826         loop .b3
827
828         ;对门进行测试
829         mov ebx,message_2
830         call far [salt_1+256]                   ;通过门显示信息(偏移量将被忽略)
831
832         mov ebx,message_3
833         call sys_routine_seg_sel:put_string ;在内核中调用例程不需要通过门
834
835         ;创建任务控制块。这不是处理器的要求，而是我们自己为了方便而设立的
836         mov ecx,0x46
837         call sys_routine_seg_sel:allocate_memory
838         call append_to_tcb_link                 ;将任务控制块追加到TCB链表
839
840         push dword 50                           ;用户程序位于逻辑50扇区
841         push ecx                                ;压入任务控制块起始线性地址
842
843         call load_relocate_program
844
845         mov ebx,do_status
846         call sys_routine_seg_sel:put_string
847
848         mov eax,mem_0_4_gb_seg_sel
```

```
849          mov ds,eax
850
851          ltr [ecx+0x18]                          ;加载任务状态段
852          lldt [ecx+0x10]                         ;加载LDT
853
854          mov eax,[ecx+0x44]
855          mov ds,eax                              ;切换到用户程序头部段
856
857          ;以下假装是从调用门返回。摹仿处理器压入返回参数
858          push dword [0x08]                       ;调用前的堆栈段选择子
859          push dword 0                            ;调用前的esp
860
861          push dword [0x14]                       ;调用前的代码段选择子
862          push dword [0x10]                       ;调用前的eip
863
864          retf
865
866 return_point:                                   ;用户程序返回点
867          mov eax,core_data_seg_sel               ;因为c14.asm是以JMP的方式使用调
868          mov ds,eax                              ;用门@TerminateProgram，回到这
869                                                  ;里时，特权级为3，会导致异常。
870          mov ebx,message_6
871          call sys_routine_seg_sel:put_string
872
873          hlt
874
875 core_code_end:
876
877 ;-------------------------------------------------------------------------------
878 SECTION core_trail
879 ;-------------------------------------------------------------------------------
880 core_end:
```