```
1           ;代码清单17-2
2           ;文件名：c17_core.asm
3           ;文件说明：保护模式微型核心程序
4           ;创建日期：2012-07-12 23:15
5  ;-------------------------------------------------------------------------------
6           ;以下定义常量
7           flat_4gb_code_seg_sel  equ  0x0008      ;平坦模型下的4GB代码段选择子
8           flat_4gb_data_seg_sel  equ  0x0018      ;平坦模型下的4GB数据段选择子
9           idt_linear_address     equ  0x8001f000  ;中断描述符表的线性基地址
10 ;-------------------------------------------------------------------------------
11          ;以下定义宏
12          %macro alloc_core_linear 0              ;在内核空间中分配虚拟内存
13                mov ebx,[core_tcb+0x06]
14                add dword [core_tcb+0x06],0x1000
15                call flat_4gb_code_seg_sel:alloc_inst_a_page
16          %endmacro
17 ;-------------------------------------------------------------------------------
18          %macro alloc_user_linear 0              ;在任务空间中分配虚拟内存
19                mov ebx,[esi+0x06]
20                add dword [esi+0x06],0x1000
21                call flat_4gb_code_seg_sel:alloc_inst_a_page
22          %endmacro
23
24 ;===============================================================================
25 SECTION  core  vstart=0x80040000
26
27          ;以下是系统核心的头部，用于加载核心程序
28          core_length      dd core_end        ;核心程序总长度#00
29
30          core_entry       dd start           ;核心代码段入口点#04
31
32 ;-------------------------------------------------------------------------------
33          [bits 32]
34 ;-------------------------------------------------------------------------------
35          ;字符串显示例程（适用于平坦内存模型）
36 put_string:                                   ;显示0终止的字符串并移动光标
37                                              ;输入：EBX=字符串的线性地址
38
39          push ebx
40          push ecx
41
42          cli                                 ;硬件操作期间，关中断
43
44    .getc:
45          mov cl,[ebx]
46          or cl,cl                            ;检测串结束标志（0）
47          jz .exit                            ;显示完毕，返回
48          call put_char
49          inc ebx
50          jmp .getc
51
52    .exit:
53
```

```
54          sti                                 ;硬件操作完毕，开放中断
55
56          pop ecx
57          pop ebx
58
59          retf                                ;段间返回
60
61 ;-------------------------------------------------------------------------------
62 put_char:                                    ;在当前光标处显示一个字符,并推进
63                                              ;光标。仅用于段内调用
64                                              ;输入：CL=字符ASCII码
65          pushad
66
67          ;以下取当前光标位置
68          mov dx,0x3d4
69          mov al,0x0e
70          out dx,al
71          inc dx                              ;0x3d5
72          in al,dx                            ;高字
73          mov ah,al
74
75          dec dx                              ;0x3d4
76          mov al,0x0f
77          out dx,al
78          inc dx                              ;0x3d5
79          in al,dx                            ;低字
80          mov bx,ax                           ;BX=代表光标位置的16位数
81          and ebx,0x0000ffff                  ;准备使用32位寻址方式访问显存
82
83          cmp cl,0x0d                         ;回车符?
84          jnz .put_0a
85
86          mov ax,bx                           ;以下按回车符处理
87          mov bl,80
88          div bl
89          mul bl
90          mov bx,ax
91          jmp .set_cursor
92
93   .put_0a:
94          cmp cl,0x0a                         ;换行符?
95          jnz .put_other
96          add bx,80                           ;增加一行
97          jmp .roll_screen
98
99   .put_other:                                ;正常显示字符
100         shl bx,1
101         mov [0x800b8000+ebx],cl             ;在光标位置处显示字符
102
103         ;以下将光标位置推进一个字符
104         shr bx,1
105         inc bx
106
```

```
107   .roll_screen:
108         cmp bx,2000                         ;光标超出屏幕？滚屏
109         jl .set_cursor
110
111         cld
112         mov esi,0x800b80a0                  ;小心！32位模式下movsb/w/d
113         mov edi,0x800b8000                  ;使用的是esi/edi/ecx
114         mov ecx,1920
115         rep movsd
116         mov bx,3840                         ;清除屏幕最底一行
117         mov ecx,80                          ;32位程序应该使用ECX
118   .cls:
119         mov word [0x800b8000+ebx],0x0720
120         add bx,2
121         loop .cls
122
123         mov bx,1920
124
125   .set_cursor:
126         mov dx,0x3d4
127         mov al,0x0e
128         out dx,al
129         inc dx                              ;0x3d5
130         mov al,bh
131         out dx,al
132         dec dx                              ;0x3d4
133         mov al,0x0f
134         out dx,al
135         inc dx                              ;0x3d5
136         mov al,bl
137         out dx,al
138
139         popad
140
141         ret
142
143 ;-------------------------------------------------------------------------------
144 read_hard_disk_0:                           ;从硬盘读取一个逻辑扇区（平坦模型）
145                                             ;EAX=逻辑扇区号
146                                             ;EBX=目标缓冲区线性地址
147                                             ;返回：EBX=EBX+512
148         cli
149
150         push eax
151         push ecx
152         push edx
153
154         push eax
155
156         mov dx,0x1f2
157         mov al,1
158         out dx,al                           ;读取的扇区数
159
```

```
160         inc dx                                  ;0x1f3
161         pop eax
162         out dx,al                               ;LBA地址7~0
163
164         inc dx                                  ;0x1f4
165         mov cl,8
166         shr eax,cl
167         out dx,al                               ;LBA地址15~8
168
169         inc dx                                  ;0x1f5
170         shr eax,cl
171         out dx,al                               ;LBA地址23~16
172
173         inc dx                                  ;0x1f6
174         shr eax,cl
175         or al,0xe0                              ;第一硬盘  LBA地址27~24
176         out dx,al
177
178         inc dx                                  ;0x1f7
179         mov al,0x20                             ;读命令
180         out dx,al
181
182   .waits:
183         in al,dx
184         and al,0x88
185         cmp al,0x08
186         jnz .waits                              ;不忙，且硬盘已准备好数据传输
187
188         mov ecx,256                             ;总共要读取的字数
189         mov dx,0x1f0
190   .readw:
191         in ax,dx
192         mov [ebx],ax
193         add ebx,2
194         loop .readw
195
196         pop edx
197         pop ecx
198         pop eax
199
200         sti
201
202         retf                                    ;远返回
203
204 ;-------------------------------------------------------------------------------
205 ;汇编语言程序是极难一次成功，而且调试非常困难。这个例程可以提供帮助
206 put_hex_dword:                                  ;在当前光标处以十六进制形式显示
207                                                 ;一个双字并推进光标
208                                                 ;输入：EDX=要转换并显示的数字
209                                                 ;输出：无
210         pushad
211
212         mov ebx,bin_hex                         ;指向核心地址空间内的转换表
```

```
213          mov ecx,8
214    .xlt:
215          rol edx,4
216          mov eax,edx
217          and eax,0x0000000f
218          xlat
219
220          push ecx
221          mov cl,al
222          call put_char
223          pop ecx
224
225          loop .xlt
226
227          popad
228          retf
229
230 ;-------------------------------------------------------------------------------
231 set_up_gdt_descriptor:                          ;在GDT内安装一个新的描述符
232                                                 ;输入：EDX:EAX=描述符
233                                                 ;输出：CX=描述符的选择子
234          push eax
235          push ebx
236          push edx
237
238          sgdt [pgdt]                            ;取得GDTR的界限和线性地址
239
240          movzx ebx,word [pgdt]                  ;GDT界限
241          inc bx                                 ;GDT总字节数，也是下一个描述符偏移
242          add ebx,[pgdt+2]                       ;下一个描述符的线性地址
243
244          mov [ebx],eax
245          mov [ebx+4],edx
246
247          add word [pgdt],8                      ;增加一个描述符的大小
248
249          lgdt [pgdt]                            ;对GDT的更改生效
250
251          mov ax,[pgdt]                          ;得到GDT界限值
252          xor dx,dx
253          mov bx,8
254          div bx                                 ;除以8，去掉余数
255          mov cx,ax
256          shl cx,3                               ;将索引号移到正确位置
257
258          pop edx
259          pop ebx
260          pop eax
261
262          retf
263 ;-------------------------------------------------------------------------------
264 make_seg_descriptor:                            ;构造存储器和系统的段描述符
265                                                 ;输入：EAX=线性基地址
```

```
266                                             ;       EBX=段界限
267                                             ;       ECX=属性。各属性位都在原始
268                                             ;           位置，无关的位清零
269                                             ;返回：EDX:EAX=描述符
270         mov edx,eax
271         shl eax,16
272         or ax,bx                            ;描述符前32位(EAX)构造完毕
273
274         and edx,0xffff0000                  ;清除基地址中无关的位
275         rol edx,8
276         bswap edx                           ;装配基址的31~24和23~16  (80486+)
277
278         xor bx,bx
279         or edx,ebx                          ;装配段界限的高4位
280
281         or edx,ecx                          ;装配属性
282
283         retf
284
285 ;----------------------------------------------------------------------------
286 make_gate_descriptor:                       ;构造门的描述符（调用门等）
287                                             ;输入：EAX=门代码在段内偏移地址
288                                             ;       BX=门代码所在段的选择子
289                                             ;       CX=段类型及属性等（各属
290                                             ;           性位都在原始位置）
291                                             ;返回：EDX:EAX=完整的描述符
292         push ebx
293         push ecx
294
295         mov edx,eax
296         and edx,0xffff0000                  ;得到偏移地址高16位
297         or dx,cx                            ;组装属性部分到EDX
298
299         and eax,0x0000ffff                  ;得到偏移地址低16位
300         shl ebx,16
301         or eax,ebx                          ;组装段选择子部分
302
303         pop ecx
304         pop ebx
305
306         retf
307
308 ;----------------------------------------------------------------------------
309 allocate_a_4k_page:                         ;分配一个4KB的页
310                                             ;输入：无
311                                             ;输出：EAX=页的物理地址
312         push ebx
313         push ecx
314         push edx
315
316         xor eax,eax
317   .b1:
318         bts [page_bit_map],eax
```

```
319         jnc .b2
320         inc eax
321         cmp eax,page_map_len*8
322         jl .b1
323
324         mov ebx,message_3
325         call flat_4gb_code_seg_sel:put_string
326         hlt                                 ;没有可以分配的页，停机
327
328   .b2:
329         shl eax,12                          ;乘以4096（0x1000）
330
331         pop edx
332         pop ecx
333         pop ebx
334
335         ret
336
337 ;-------------------------------------------------------------------------------
338 alloc_inst_a_page:                          ;分配一个页，并安装在当前活动的
339                                             ;层级分页结构中
340                                             ;输入：EBX=页的线性地址
341         push eax
342         push ebx
343         push esi
344
345         ;检查该线性地址所对应的页表是否存在
346         mov esi,ebx
347         and esi,0xffc00000
348         shr esi,20                          ;得到页目录索引，并乘以4
349         or esi,0xfffff000                   ;页目录自身的线性地址+表内偏移
350
351         test dword [esi],0x00000001         ;P位是否为“1”。检查该线性地址是
352         jnz .b1                             ;否已经有对应的页表
353
354         ;创建该线性地址所对应的页表
355         call allocate_a_4k_page             ;分配一个页做为页表
356         or eax,0x00000007
357         mov [esi],eax                       ;在页目录中登记该页表
358
359   .b1:
360         ;开始访问该线性地址所对应的页表
361         mov esi,ebx
362         shr esi,10
363         and esi,0x003ff000                  ;或者0xfffff000，因高10位是零
364         or esi,0xffc00000                   ;得到该页表的线性地址
365
366         ;得到该线性地址在页表内的对应条目（页表项）
367         and ebx,0x003ff000
368         shr ebx,10                          ;相当于右移12位，再乘以4
369         or esi,ebx                          ;页表项的线性地址
370         call allocate_a_4k_page             ;分配一个页，这才是要安装的页
371         or eax,0x00000007
```

```
372          mov [esi],eax
373
374          pop esi
375          pop ebx
376          pop eax
377
378          retf
379
380 ;------------------------------------------------------------------------------
381 create_copy_cur_pdir:                           ;创建新页目录，并复制当前页目录内容
382                                                 ;输入：无
383                                                 ;输出：EAX=新页目录的物理地址
384          push esi
385          push edi
386          push ebx
387          push ecx
388
389          call allocate_a_4k_page
390          mov ebx,eax
391          or ebx,0x00000007
392          mov [0xfffffff8],ebx
393
394          invlpg [0xfffffff8]
395
396          mov esi,0xfffff000                     ;ESI->当前页目录的线性地址
397          mov edi,0xffffe000                     ;EDI->新页目录的线性地址
398          mov ecx,1024                           ;ECX=要复制的目录项数
399          cld
400          repe movsd
401
402          pop ecx
403          pop ebx
404          pop edi
405          pop esi
406
407          retf
408
409 ;------------------------------------------------------------------------------
410 general_interrupt_handler:                      ;通用的中断处理过程
411          push eax
412
413          mov al,0x20                            ;中断结束命令EOI
414          out 0xa0,al                            ;向从片发送
415          out 0x20,al                            ;向主片发送
416
417          pop eax
418
419          iretd
420
421 ;------------------------------------------------------------------------------
422 general_exception_handler:                      ;通用的异常处理过程
423          mov ebx,excep_msg
424          call flat_4gb_code_seg_sel:put_string
```

```
425
426         hlt
427
428 ;-------------------------------------------------------------------------------
429 rtm_0x70_interrupt_handle:                  ;实时时钟中断处理过程
430
431         pushad
432
433         mov al,0x20                         ;中断结束命令EOI
434         out 0xa0,al                         ;向8259A从片发送
435         out 0x20,al                         ;向8259A主片发送
436
437         mov al,0x0c                         ;寄存器C的索引。且开放NMI
438         out 0x70,al
439         in al,0x71                          ;读一下RTC的寄存器C，否则只发生一次中断
440                                             ;此处不考虑闹钟和周期性中断的情况
441         ;找当前任务（状态为忙的任务）在链表中的位置
442         mov eax,tcb_chain
443   .b0:                                      ;EAX=链表头或当前TCB线性地址
444         mov ebx,[eax]                       ;EBX=下一个TCB线性地址
445         or ebx,ebx
446         jz .irtn                            ;链表为空，或已到末尾，从中断返回
447         cmp word [ebx+0x04],0xffff          ;是忙任务（当前任务）？
448         je .b1
449         mov eax,ebx                         ;定位到下一个TCB（的线性地址）
450         jmp .b0
451
452         ;将当前为忙的任务移到链尾
453   .b1:
454         mov ecx,[ebx]                       ;下游TCB的线性地址
455         mov [eax],ecx                       ;将当前任务从链中拆除
456
457   .b2:                                      ;此时，EBX=当前任务的线性地址
458         mov edx,[eax]
459         or edx,edx                          ;已到链表尾端？
460         jz .b3
461         mov eax,edx
462         jmp .b2
463
464   .b3:
465         mov [eax],ebx                       ;将忙任务的TCB挂在链表尾端
466         mov dword [ebx],0x00000000          ;将忙任务的TCB标记为链尾
467
468         ;从链首搜索第一个空闲任务
469         mov eax,tcb_chain
470   .b4:
471         mov eax,[eax]
472         or eax,eax                          ;已到链尾（未发现空闲任务）
473         jz .irtn                            ;未发现空闲任务，从中断返回
474         cmp word [eax+0x04],0x0000          ;是空闲任务？
475         jnz .b4
476
477         ;将空闲任务和当前任务的状态都取反
```

```
478         not word [eax+0x04]                 ;设置空闲任务的状态为忙
479         not word [ebx+0x04]                 ;设置当前任务（忙）的状态为空闲
480         jmp far [eax+0x14]                  ;任务转换
481
482   .irtn:
483         popad
484
485         iretd
486
487 ;-------------------------------------------------------------------------------
488 terminate_current_task:                     ;终止当前任务
489                                             ;注意，执行此例程时，当前任务仍在
490                                             ;运行中。此例程其实也是当前任务的
491                                             ;一部分
492         ;找当前任务（状态为忙的任务）在链表中的位置
493         mov eax,tcb_chain
494   .b0:                                      ;EAX=链表头或当前TCB线性地址
495         mov ebx,[eax]                       ;EBX=下一个TCB线性地址
496         cmp word [ebx+0x04],0xffff          ;是忙任务（当前任务）？
497         je .b1
498         mov eax,ebx                         ;定位到下一个TCB（的线性地址）
499         jmp .b0
500
501   .b1:
502         mov word [ebx+0x04],0x3333          ;修改当前任务的状态为"退出"
503
504   .b2:
505         hlt                                 ;停机，等待程序管理器恢复运行时，
506                                             ;将其回收
507         jmp .b2
508
509 ;-------------------------------------------------------------------------------
510         pgdt             dw  0              ;用于设置和修改GDT
511                          dd  0
512
513         pidt             dw  0
514                          dd  0
515
516         ;任务控制块链
517         tcb_chain        dd  0
518
519         core_tcb   times  32  db 0          ;内核（程序管理器）的TCB
520
521         page_bit_map     db  0xff,0xff,0xff,0xff,0xff,0xff,0x55,0x55
522                          db  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
523                          db  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
524                          db  0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
525                          db  0x55,0x55,0x55,0x55,0x55,0x55,0x55,0x55
526                          db  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
527                          db  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
528                          db  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
529         page_map_len     equ $-page_bit_map
530
```

```
531         ;符号地址检索表
532         salt:
533         salt_1            db  '@PrintString'
534                       times 256-($-salt_1) db 0
535                           dd  put_string
536                           dw  flat_4gb_code_seg_sel
537
538         salt_2            db  '@ReadDiskData'
539                       times 256-($-salt_2) db 0
540                           dd  read_hard_disk_0
541                           dw  flat_4gb_code_seg_sel
542
543         salt_3            db  '@PrintDwordAsHexString'
544                       times 256-($-salt_3) db 0
545                           dd  put_hex_dword
546                           dw  flat_4gb_code_seg_sel
547
548         salt_4            db  '@TerminateProgram'
549                       times 256-($-salt_4) db 0
550                           dd  terminate_current_task
551                           dw  flat_4gb_code_seg_sel
552
553         salt_item_len   equ $-salt_4
554         salt_items      equ ($-salt)/salt_item_len
555
556         excep_msg         db  '********Exception encounted********',0
557
558         message_0         db  '  Working in system core with protection '
559                           db  'and paging are all enabled.System core is mapped '
560                           db  'to address 0x80000000.',0x0d,0x0a,0
561
562         message_1         db  '  System wide CALL-GATE mounted.',0x0d,0x0a,0
563
564         message_3         db  '********No more pages********',0
565
566         core_msg0         db  '  System core task running!',0x0d,0x0a,0
567
568         bin_hex           db '0123456789ABCDEF'
569                                               ;put_hex_dword子过程用的查找表
570
571         core_buf   times 512 db 0             ;内核用的缓冲区
572
573         cpu_brnd0         db 0x0d,0x0a,'  ',0
574         cpu_brand  times 52 db 0
575         cpu_brnd1         db 0x0d,0x0a,0x0d,0x0a,0
576
577 ;-------------------------------------------------------------------------------
578 fill_descriptor_in_ldt:                       ;在LDT内安装一个新的描述符
579                                               ;输入：EDX:EAX=描述符
580                                               ;           EBX=TCB基地址
581                                               ;输出：CX=描述符的选择子
582         push eax
583         push edx
```

```
584         push edi
585
586         mov edi,[ebx+0x0c]                  ;获得LDT基地址
587
588         xor ecx,ecx
589         mov cx,[ebx+0x0a]                   ;获得LDT界限
590         inc cx                              ;LDT的总字节数，即新描述符偏移地址
591
592         mov [edi+ecx+0x00],eax
593         mov [edi+ecx+0x04],edx              ;安装描述符
594
595         add cx,8
596         dec cx                              ;得到新的LDT界限值
597
598         mov [ebx+0x0a],cx                   ;更新LDT界限值到TCB
599
600         mov ax,cx
601         xor dx,dx
602         mov cx,8
603         div cx
604
605         mov cx,ax
606         shl cx,3                            ;左移3位，并且
607         or cx,0000_0000_0000_0100B          ;使TI位=1，指向LDT，最后使RPL=00
608
609         pop edi
610         pop edx
611         pop eax
612
613         ret
614
615 ;-------------------------------------------------------------------------------
616 load_relocate_program:                      ;加载并重定位用户程序
617                                             ;输入: PUSH 逻辑扇区号
618                                             ;       PUSH 任务控制块基地址
619                                             ;输出：无
620         pushad
621
622         mov ebp,esp                         ;为访问通过堆栈传递的参数做准备
623
624         ;清空当前页目录的前半部分（对应低2GB的局部地址空间）
625         mov ebx,0xfffff000
626         xor esi,esi
627   .b1:
628         mov dword [ebx+esi*4],0x00000000
629         inc esi
630         cmp esi,512
631         jl .b1
632
633         mov eax,cr3
634         mov cr3,eax                         ;刷新TLB
635
636         ;以下开始分配内存并加载用户程序
```

```
637         mov eax,[ebp+40]                    ;从堆栈中取出用户程序起始扇区号
638         mov ebx,core_buf                    ;读取程序头部数据
639         call flat_4gb_code_seg_sel:read_hard_disk_0
640
641         ;以下判断整个程序有多大
642         mov eax,[core_buf]                  ;程序尺寸
643         mov ebx,eax
644         and ebx,0xfffff000                  ;使之4KB对齐
645         add ebx,0x1000
646         test eax,0x00000fff                 ;程序的大小正好是4KB的倍数吗?
647         cmovnz eax,ebx                      ;不是。使用凑整的结果
648
649         mov ecx,eax
650         shr ecx,12                          ;程序占用的总4KB页数
651
652         mov eax,[ebp+40]                    ;起始扇区号
653         mov esi,[ebp+36]                    ;从堆栈中取得TCB的基地址
654   .b2:
655         alloc_user_linear                   ;宏：在用户任务地址空间上分配内存
656
657         push ecx
658         mov ecx,8
659   .b3:
660         call flat_4gb_code_seg_sel:read_hard_disk_0
661         inc eax
662         loop .b3
663
664         pop ecx
665         loop .b2
666
667         ;在内核地址空间内创建用户任务的TSS
668         alloc_core_linear                   ;宏：在内核的地址空间上分配内存
669                                             ;用户任务的TSS必须在全局空间上分配
670
671         mov [esi+0x14],ebx                  ;在TCB中填写TSS的线性地址
672         mov word [esi+0x12],103             ;在TCB中填写TSS的界限值
673
674         ;在用户任务的局部地址空间内创建LDT
675         alloc_user_linear                   ;宏：在用户任务地址空间上分配内存
676
677         mov [esi+0x0c],ebx                  ;填写LDT线性地址到TCB中
678
679         ;建立程序代码段描述符
680         mov eax,0x00000000
681         mov ebx,0x000fffff
682         mov ecx,0x00c0f800                  ;4KB粒度的代码段描述符，特权级3
683         call flat_4gb_code_seg_sel:make_seg_descriptor
684         mov ebx,esi                         ;TCB的基地址
685         call fill_descriptor_in_ldt
686         or cx,0000_0000_0000_0011B          ;设置选择子的特权级为3
687
688         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
689         mov [ebx+76],cx                     ;填写TSS的CS域
```

```
690
691         ;建立程序数据段描述符
692         mov eax,0x00000000
693         mov ebx,0x000fffff
694         mov ecx,0x00c0f200                  ;4KB粒度的数据段描述符，特权级3
695         call flat_4gb_code_seg_sel:make_seg_descriptor
696         mov ebx,esi                         ;TCB的基地址
697         call fill_descriptor_in_ldt
698         or cx,0000_0000_0000_0011B          ;设置选择子的特权级为3
699
700         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
701         mov [ebx+84],cx                     ;填写TSS的DS域
702         mov [ebx+72],cx                     ;填写TSS的ES域
703         mov [ebx+88],cx                     ;填写TSS的FS域
704         mov [ebx+92],cx                     ;填写TSS的GS域
705
706         ;将数据段作为用户任务的3特权级固有堆栈
707         alloc_user_linear                   ;宏：在用户任务地址空间上分配内存
708
709         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
710         mov [ebx+80],cx                     ;填写TSS的SS域
711         mov edx,[esi+0x06]                  ;堆栈的高端线性地址
712         mov [ebx+56],edx                    ;填写TSS的ESP域
713
714         ;在用户任务的局部地址空间内创建0特权级堆栈
715         alloc_user_linear                   ;宏：在用户任务地址空间上分配内存
716
717         mov eax,0x00000000
718         mov ebx,0x000fffff
719         mov ecx,0x00c09200                  ;4KB粒度的堆栈段描述符，特权级0
720         call flat_4gb_code_seg_sel:make_seg_descriptor
721         mov ebx,esi                         ;TCB的基地址
722         call fill_descriptor_in_ldt
723         or cx,0000_0000_0000_0000B          ;设置选择子的特权级为0
724
725         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
726         mov [ebx+8],cx                      ;填写TSS的SS0域
727         mov edx,[esi+0x06]                  ;堆栈的高端线性地址
728         mov [ebx+4],edx                     ;填写TSS的ESP0域
729
730         ;在用户任务的局部地址空间内创建1特权级堆栈
731         alloc_user_linear                   ;宏：在用户任务地址空间上分配内存
732
733         mov eax,0x00000000
734         mov ebx,0x000fffff
735         mov ecx,0x00c0b200                  ;4KB粒度的堆栈段描述符，特权级1
736         call flat_4gb_code_seg_sel:make_seg_descriptor
737         mov ebx,esi                         ;TCB的基地址
738         call fill_descriptor_in_ldt
739         or cx,0000_0000_0000_0001B          ;设置选择子的特权级为1
740
741         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
742         mov [ebx+16],cx                     ;填写TSS的SS1域
```

```
743         mov edx,[esi+0x06]                  ;堆栈的高端线性地址
744         mov [ebx+12],edx                    ;填写TSS的ESP1域
745
746         ;在用户任务的局部地址空间内创建2特权级堆栈
747         alloc_user_linear                   ;宏：在用户任务地址空间上分配内存
748
749         mov eax,0x00000000
750         mov ebx,0x000fffff
751         mov ecx,0x00c0d200                  ;4KB粒度的堆栈段描述符，特权级2
752         call flat_4gb_code_seg_sel:make_seg_descriptor
753         mov ebx,esi                         ;TCB的基地址
754         call fill_descriptor_in_ldt
755         or cx,0000_0000_0000_0010B          ;设置选择子的特权级为2
756
757         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
758         mov [ebx+24],cx                     ;填写TSS的SS2域
759         mov edx,[esi+0x06]                  ;堆栈的高端线性地址
760         mov [ebx+20],edx                    ;填写TSS的ESP2域
761
762         ;重定位U-SALT
763         cld
764
765         mov ecx,[0x0c]                      ;U-SALT条目数
766         mov edi,[0x08]                      ;U-SALT在4GB空间内的偏移
767   .b4:
768         push ecx
769         push edi
770
771         mov ecx,salt_items
772         mov esi,salt
773   .b5:
774         push edi
775         push esi
776         push ecx
777
778         mov ecx,64                          ;检索表中，每条目的比较次数
779         repe cmpsd                          ;每次比较4字节
780         jnz .b6
781         mov eax,[esi]                       ;若匹配，则esi恰好指向其后的地址
782         mov [edi-256],eax                   ;将字符串改写成偏移地址
783         mov ax,[esi+4]
784         or ax,0000000000000011B             ;以用户程序自己的特权级使用调用门
785                                             ;故RPL=3
786         mov [edi-252],ax                    ;回填调用门选择子
787   .b6:
788
789         pop ecx
790         pop esi
791         add esi,salt_item_len
792         pop edi                             ;从头比较
793         loop .b5
794
795         pop edi
```

```
796         add edi,256
797         pop ecx
798         loop .b4
799
800         ;在GDT中登记LDT描述符
801         mov esi,[ebp+36]                    ;从堆栈中取得TCB的基地址
802         mov eax,[esi+0x0c]                  ;LDT的起始线性地址
803         movzx ebx,word [esi+0x0a]           ;LDT段界限
804         mov ecx,0x00408200                  ;LDT描述符，特权级0
805         call flat_4gb_code_seg_sel:make_seg_descriptor
806         call flat_4gb_code_seg_sel:set_up_gdt_descriptor
807         mov [esi+0x10],cx                   ;登记LDT选择子到TCB中
808
809         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
810         mov [ebx+96],cx                     ;填写TSS的LDT域
811
812         mov word [ebx+0],0                  ;反向链=0
813
814         mov dx,[esi+0x12]                   ;段长度（界限）
815         mov [ebx+102],dx                    ;填写TSS的I/O位图偏移域
816
817         mov word [ebx+100],0                ;T=0
818
819         mov eax,[0x04]                      ;从任务的4GB地址空间获取入口点
820         mov [ebx+32],eax                    ;填写TSS的EIP域
821
822         pushfd
823         pop edx
824         mov [ebx+36],edx                    ;填写TSS的EFLAGS域
825
826         ;在GDT中登记TSS描述符
827         mov eax,[esi+0x14]                  ;从TCB中获取TSS的起始线性地址
828         movzx ebx,word [esi+0x12]           ;段长度（界限）
829         mov ecx,0x00408900                  ;TSS描述符，特权级0
830         call flat_4gb_code_seg_sel:make_seg_descriptor
831         call flat_4gb_code_seg_sel:set_up_gdt_descriptor
832         mov [esi+0x18],cx                   ;登记TSS选择子到TCB
833
834         ;创建用户任务的页目录
835         ;注意！页的分配和使用是由页位图决定的，可以不占用线性地址空间
836         call flat_4gb_code_seg_sel:create_copy_cur_pdir
837         mov ebx,[esi+0x14]                  ;从TCB中获取TSS的线性地址
838         mov dword [ebx+28],eax              ;填写TSS的CR3(PDBR)域
839
840         popad
841
842         ret 8                               ;丢弃调用本过程前压入的参数
843
844 ;-------------------------------------------------------------------------------
845 append_to_tcb_link:                         ;在TCB链上追加任务控制块
846                                             ;输入：ECX=TCB线性基地址
847         cli
848
```

```
849         push eax
850         push ebx
851
852         mov eax,tcb_chain
853   .b0:                                          ;EAX=链表头或当前TCB线性地址
854         mov ebx,[eax]                         ;EBX=下一个TCB线性地址
855         or ebx,ebx
856         jz .b1                                ;链表为空，或已到末尾
857         mov eax,ebx                           ;定位到下一个TCB（的线性地址）
858         jmp .b0
859
860   .b1:
861         mov [eax],ecx
862         mov dword [ecx],0x00000000            ;当前TCB指针域清零，以指示这是最
863                                               ;后一个TCB
864         pop ebx
865         pop eax
866
867         sti
868
869         ret
870
871 ;-------------------------------------------------------------------------------
872 start:
873         ;创建中断描述符表IDT
874         ;在此之前，禁止调用put_string过程，以及任何含有sti指令的过程。
875
876         ;前20个向量是处理器异常使用的
877         mov eax,general_exception_handler     ;门代码在段内偏移地址
878         mov bx,flat_4gb_code_seg_sel          ;门代码所在段的选择子
879         mov cx,0x8e00                         ;32位中断门，0特权级
880         call flat_4gb_code_seg_sel:make_gate_descriptor
881
882         mov ebx,idt_linear_address            ;中断描述符表的线性地址
883         xor esi,esi
884   .idt0:
885         mov [ebx+esi*8],eax
886         mov [ebx+esi*8+4],edx
887         inc esi
888         cmp esi,19                            ;安装前20个异常中断处理过程
889         jle .idt0
890
891         ;其余为保留或硬件使用的中断向量
892         mov eax,general_interrupt_handler     ;门代码在段内偏移地址
893         mov bx,flat_4gb_code_seg_sel          ;门代码所在段的选择子
894         mov cx,0x8e00                         ;32位中断门，0特权级
895         call flat_4gb_code_seg_sel:make_gate_descriptor
896
897         mov ebx,idt_linear_address            ;中断描述符表的线性地址
898   .idt1:
899         mov [ebx+esi*8],eax
900         mov [ebx+esi*8+4],edx
901         inc esi
```

```
902         cmp esi,255                             ;安装普通的中断处理过程
903         jle .idt1
904
905         ;设置实时时钟中断处理过程
906         mov eax,rtm_0x70_interrupt_handle       ;门代码在段内偏移地址
907         mov bx,flat_4gb_code_seg_sel            ;门代码所在段的选择子
908         mov cx,0x8e00                           ;32位中断门，0特权级
909         call flat_4gb_code_seg_sel:make_gate_descriptor
910
911         mov ebx,idt_linear_address              ;中断描述符表的线性地址
912         mov [ebx+0x70*8],eax
913         mov [ebx+0x70*8+4],edx
914
915         ;准备开放中断
916         mov word [pidt],256*8-1                 ;IDT的界限
917         mov dword [pidt+2],idt_linear_address
918         lidt [pidt]                             ;加载中断描述符表寄存器IDTR
919
920         ;设置8259A中断控制器
921         mov al,0x11
922         out 0x20,al                             ;ICW1：边沿触发/级联方式
923         mov al,0x20
924         out 0x21,al                             ;ICW2:起始中断向量
925         mov al,0x04
926         out 0x21,al                             ;ICW3:从片级联到IR2
927         mov al,0x01
928         out 0x21,al                             ;ICW4:非总线缓冲，全嵌套，正常EOI
929
930         mov al,0x11
931         out 0xa0,al                             ;ICW1：边沿触发/级联方式
932         mov al,0x70
933         out 0xa1,al                             ;ICW2:起始中断向量
934         mov al,0x04
935         out 0xa1,al                             ;ICW3:从片级联到IR2
936         mov al,0x01
937         out 0xa1,al                             ;ICW4:非总线缓冲，全嵌套，正常EOI
938
939         ;设置和时钟中断相关的硬件
940         mov al,0x0b                             ;RTC寄存器B
941         or al,0x80                              ;阻断NMI
942         out 0x70,al
943         mov al,0x12                             ;设置寄存器B，禁止周期性中断，开放更
944         out 0x71,al                             ;新结束后中断，BCD码，24小时制
945
946         in al,0xa1                              ;读8259从片的IMR寄存器
947         and al,0xfe                             ;清除bit 0(此位连接RTC)
948         out 0xa1,al                             ;写回此寄存器
949
950         mov al,0x0c
951         out 0x70,al
952         in al,0x71                              ;读RTC寄存器C，复位未决的中断状态
953
954         sti                                     ;开放硬件中断
```

```
955
956         mov ebx,message_0
957         call flat_4gb_code_seg_sel:put_string
958
959         ;显示处理器品牌信息
960         mov eax,0x80000002
961         cpuid
962         mov [cpu_brand + 0x00],eax
963         mov [cpu_brand + 0x04],ebx
964         mov [cpu_brand + 0x08],ecx
965         mov [cpu_brand + 0x0c],edx
966
967         mov eax,0x80000003
968         cpuid
969         mov [cpu_brand + 0x10],eax
970         mov [cpu_brand + 0x14],ebx
971         mov [cpu_brand + 0x18],ecx
972         mov [cpu_brand + 0x1c],edx
973
974         mov eax,0x80000004
975         cpuid
976         mov [cpu_brand + 0x20],eax
977         mov [cpu_brand + 0x24],ebx
978         mov [cpu_brand + 0x28],ecx
979         mov [cpu_brand + 0x2c],edx
980
981         mov ebx,cpu_brnd0                       ;显示处理器品牌信息
982         call flat_4gb_code_seg_sel:put_string
983         mov ebx,cpu_brand
984         call flat_4gb_code_seg_sel:put_string
985         mov ebx,cpu_brnd1
986         call flat_4gb_code_seg_sel:put_string
987
988         ;以下开始安装为整个系统服务的调用门。特权级之间的控制转移必须使用门
989         mov edi,salt                            ;C-SALT表的起始位置
990         mov ecx,salt_items                      ;C-SALT表的条目数量
991   .b4:
992         push ecx
993         mov eax,[edi+256]                       ;该条目入口点的32位偏移地址
994         mov bx,[edi+260]                        ;该条目入口点的段选择子
995         mov cx,1_11_0_1100_000_00000B           ;特权级3的调用门(3以上的特权级才
996                                                 ;允许访问)，0个参数(因为用寄存器
997                                                 ;传递参数，而没有用栈)
998         call flat_4gb_code_seg_sel:make_gate_descriptor
999         call flat_4gb_code_seg_sel:set_up_gdt_descriptor
1000        mov [edi+260],cx                        ;将返回的门描述符选择子回填
1001        add edi,salt_item_len                   ;指向下一个C-SALT条目
1002        pop ecx
1003        loop .b4
1004
1005        ;对门进行测试
1006        mov ebx,message_1
1007        call far [salt_1+256]                   ;通过门显示信息(偏移量将被忽略)
```

```
1008
1009        ;初始化创建程序管理器任务的任务控制块TCB
1010        mov word [core_tcb+0x04],0xffff     ;任务状态：忙碌
1011        mov dword [core_tcb+0x06],0x80100000
1012                                            ;内核虚拟空间的分配从这里开始。
1013        mov word [core_tcb+0x0a],0xffff     ;登记LDT初始的界限到TCB中（未使用）
1014        mov ecx,core_tcb
1015        call append_to_tcb_link             ;将此TCB添加到TCB链中
1016
1017        ;为程序管理器的TSS分配内存空间
1018        alloc_core_linear                   ;宏：在内核的虚拟地址空间分配内存
1019
1020        ;在程序管理器的TSS中设置必要的项目
1021        mov word [ebx+0],0                  ;反向链=0
1022        mov eax,cr3
1023        mov dword [ebx+28],eax              ;登记CR3(PDBR)
1024        mov word [ebx+96],0                 ;没有LDT。处理器允许没有LDT的任务。
1025        mov word [ebx+100],0                ;T=0
1026        mov word [ebx+102],103              ;没有I/O位图。0特权级事实上不需要。
1027
1028        ;创建程序管理器的TSS描述符，并安装到GDT中
1029        mov eax,ebx                         ;TSS的起始线性地址
1030        mov ebx,103                         ;段长度（界限）
1031        mov ecx,0x00408900                  ;TSS描述符，特权级0
1032        call flat_4gb_code_seg_sel:make_seg_descriptor
1033        call flat_4gb_code_seg_sel:set_up_gdt_descriptor
1034        mov [core_tcb+0x18],cx              ;登记内核任务的TSS选择子到其TCB
1035
1036        ;任务寄存器TR中的内容是任务存在的标志，该内容也决定了当前任务是谁。
1037        ;下面的指令为当前正在执行的0特权级任务“程序管理器”后补手续（TSS）。
1038        ltr cx
1039
1040        ;现在可认为“程序管理器”任务正执行中
1041
1042        ;创建用户任务的任务控制块
1043        alloc_core_linear                   ;宏：在内核的虚拟地址空间分配内存
1044
1045        mov word [ebx+0x04],0               ;任务状态：空闲
1046        mov dword [ebx+0x06],0              ;用户任务局部空间的分配从0开始。
1047        mov word [ebx+0x0a],0xffff          ;登记LDT初始的界限到TCB中
1048
1049        push dword 50                       ;用户程序位于逻辑50扇区
1050        push ebx                            ;压入任务控制块起始线性地址
1051        call load_relocate_program
1052        mov ecx,ebx
1053        call append_to_tcb_link             ;将此TCB添加到TCB链中
1054
1055        ;创建用户任务的任务控制块
1056        alloc_core_linear                   ;宏：在内核的虚拟地址空间分配内存
1057
1058        mov word [ebx+0x04],0               ;任务状态：空闲
1059        mov dword [ebx+0x06],0              ;用户任务局部空间的分配从0开始。
1060        mov word [ebx+0x0a],0xffff          ;登记LDT初始的界限到TCB中
```

```
1061
1062          push dword 100                        ;用户程序位于逻辑100扇区
1063          push ebx                              ;压入任务控制块起始线性地址
1064          call load_relocate_program
1065          mov ecx,ebx
1066          call append_to_tcb_link               ;将此TCB添加到TCB链中
1067
1068    .core:
1069          mov ebx,core_msg0
1070          call flat_4gb_code_seg_sel:put_string
1071
1072          ;这里可以编写回收已终止任务内存的代码
1073
1074          jmp .core
1075
1076 core_code_end:
1077
1078 ;-------------------------------------------------------------------------------
1079 SECTION core_trail
1080 ;-------------------------------------------------------------------------------
1081 core_end:
```