## 程序 10-1 linux/kernel/chr_drv/keyboard.S

```
/*
 *  linux/kernel/keyboard.S
 *
 *  (C) 1991  Linus Torvalds
 */

/*
 *      Thanks to Alfred Leung for US keyboard patches
 *              Wolfgang Thiel for German keyboard patches
 *              Marc Corsini for the French keyboard
 */
/*
 *      感谢 Alfred Leung 添加了 US 键盘补丁程序；
 *              Wolfgang Thiel 添加了德语键盘补丁程序；
 *              Marc Corsini 添加了法文键盘补丁程序。
 */

/* KBD_FINNISH for Finnish keyboards
 * KBD_US for US-type
 * KBD_GR for German keyboards
 * KBD_FR for Frech keyboard
 */
/*
 * KBD_FINNISH 是芬兰键盘。
 * KBD_US 是美式键盘。
 * KBD_GR 是德式键盘。
 * KBD_FR 是法式键盘。
 */
#define KBD_FINNISH         // 定义使用的键盘类型。用于后面选择采用的字符映射码表。

.text
.globl _keyboard_interrupt  // 申明为全局变量，用于在初始化时设置键盘中断描述符。

/*
 * these are for the keyboard read functions
 */
/*
 * 以下这些用于读键盘操作。
 */
// size 是键盘缓冲区（缓冲队列）长度（字节数）。
/* 值必须是 2 的次方! 并且与 tty_io.c 中的值匹配!!!! */
size    = 1024          /* must be a power of two ! And MUST be the same
                           as in tty_io.c !!!! */

// 以下是键盘缓冲队列数据结构 tty_queue 中的偏移量（include/linux/tty.h，第 16 行）。
head = 4                // 缓冲区头指针字段在 tty_queue 结构中的偏移。
tail = 8                // 缓冲区尾指针字段偏移。
proc_list = 12          // 等待该缓冲队列的进程字段偏移。
buf = 16                // 缓冲区字段偏移。

// 在本程序中使用了 3 个标志字节。mode 是键盘特殊键（ctrl、alt 或 caps）的按下状态标志；
// leds 是用于表示键盘指示灯的状态标志。 e0 是当收到扫描码 0xe0 或 0xe1 时设置的标志。
```

```
// 每个字节标志中各位的含义见如下说明：
// (1) mode 是键盘特殊键的按下状态标志。
// 表示大小写转换键(caps)、交换键(alt)、控制键(ctrl)和换档键(shift)的状态。
//      位 7 caps 键按下；
//      位 6 caps 键的状态（应该与 leds 中对应 caps 的标志位一样）；
//      位 5 右 alt 键按下；
//      位 4 左 alt 键按下；
//      位 3 右 ctrl 键按下；
//      位 2 左 ctrl 键按下；
//      位 1 右 shift 键按下；
//      位 0 左 shift 键按下。
// (2) leds 是用于表示键盘指示灯的状态标志。即表示数字锁定键（num-lock）、大小写转换
// 键（caps-lock）和滚动锁定键（scroll-lock）的 LED 发光管状态。
//      位 7-3 全 0 不用；
//      位 2 caps-lock；
//      位 1 num-lock（初始置 1，也即设置数字锁定键(num-lock)发光管为亮）；
//      位 0 scroll-lock。
// (3) 当扫描码是 0xe0 或 0xe1 时，置该标志。表示其后还跟随着 1 个或 2 个字符扫描码。通
// 常若收到扫描码 0xe0 则意味着还有一个字符跟随其后；若收到扫描码 0xe1 则表示后面还跟
// 随着 2 个字符。参见程序列表后说明。
//      位 1 =1 收到 0xe1 标志；
//      位 0 =1 收到 0xe0 标志。
33 mode:   .byte 0          /* caps, alt, ctrl and shift mode */
34 leds:   .byte 2          /* num-lock, caps, scroll-lock mode (nom-lock on) */
35 e0:     .byte 0
36
37 /*
38  *  con_int is the real interrupt routine that reads the
39  *  keyboard scan-code and converts it into the appropriate
40  *  ascii character(s).
41  */
   /*
    * con_int 是实际的中断处理子程序，用于读键盘扫描码并将其转换
    * 成相应的 ascii 字符。[ 注：这段英文注释已过时。]
    */
   //// 键盘中断处理程序入口点。
   // 当键盘控制器接收到用户的一个按键操作时，就会向中断控制器发出一个键盘中断请求信号
   // IRQ1。当 CPU 响应该请求时就会执行键盘中断处理程序。该中断处理程序会从键盘控制器相
   // 应端口（0x60）读入按键扫描码，并调用对应的扫描码子程序进行处理。
   // 首先从端口 0x60 读取当前按键的扫描码。然后判断该扫描码是否是 0xe0 或 0xe1，如果是
   // 的话就立刻对键盘控制器作出应答，并向中断控制器发送中断结束（EOI）信号，以允许键
   // 盘控制器能继续产生中断信号，从而让我们来接收后续的字符。 如果接收到的扫描码不是
   // 这两个特殊扫描码，我们就根据扫描码值调用按键跳转表 key_table 中相应按键处理子程
   // 序，把扫描码对应的字符放入读字符缓冲队列 read_q 中。然后，在对键盘控制器作出应答
   // 并发送 EOI 信号之后，调用函数 do_tty_interrupt()（实际上是调用 copy_to_cooked()）
   // 把 read_q 中的字符经过处理后放到 secondary 辅助队列中。
42 _keyboard_interrupt:
43         pushl %eax
44         pushl %ebx
45         pushl %ecx
46         pushl %edx
47         push %ds
48         push %es
```

```
49          movl $0x10,%eax         // 将 ds、es 段寄存器置为内核数据段。
50          mov %ax,%ds
51          mov %ax,%es
52          movl _blankinterval,%eax
53          movl %eax,_blankcount   // 预置黑屏时间计数值为 blankinterval（滴答数）。
54          xorl %eax,%eax          /* %eax is scan code */    /* eax 中是扫描码 */
55          inb $0x60,%al           // 读取扫描码➔al。
56          cmpb $0xe0,%al          // 扫描码是 0xe0 吗？若是则跳转到设置 e0 标志代码处。
57          je set_e0
58          cmpb $0xe1,%al          // 扫描码是 0xe1 吗？若是则跳转到设置 e1 标志代码处。
59          je set_e1
60          call key_table(,%eax,4) // 调用键处理程序 key_table + eax*4（参见 502 行）。
61          movb $0,e0              // 返回之后复位 e0 标志。

    // 下面这段代码（55-65 行）针对使用 8255A 的 PC 标准键盘电路进行硬件复位处理。端口 0x61
    // 是 8255A 输出口 B 的地址，该输出端口的第 7 位（PB7）用于禁止和允许对键盘数据的处理。
    // 这段程序用于对收到的扫描码做出应答。方法是首先禁止键盘，然后立刻重新允许键盘工作。
62  e0_e1:  inb $0x61,%al           // 取 PPI 端口 B 状态，其位 7 用于允许/禁止（0/1）键盘。
63          jmp 1f                  // 延迟一会。
64  1:      jmp 1f
65  1:      orb $0x80,%al           // al 位 7 置位（禁止键盘工作）。
66          jmp 1f
67  1:      jmp 1f
68  1:      outb %al,$0x61          // 使 PPI PB7 位置位。
69          jmp 1f
70  1:      jmp 1f
71  1:      andb $0x7F,%al          // al 位 7 复位。
72          outb %al,$0x61          // 使 PPI PB7 位复位（允许键盘工作）。
73          movb $0x20,%al          // 向 8259 中断芯片发送 EOI(中断结束)信号。
74          outb %al,$0x20
75          pushl $0                // 控制台 tty 号=0，作为参数入栈。
76          call _do_tty_interrupt  // 将收到数据转换成规范模式并存放在规范字符缓冲队列中。
77          addl $4,%esp            // 丢弃入栈的参数，弹出保留的寄存器，并中断返回。
78          pop %es
79          pop %ds
80          popl %edx
81          popl %ecx
82          popl %ebx
83          popl %eax
84          iret
85  set_e0: movb $1,e0              // 收到扫描前导码 0xe0 时，设置 e0 标志（位 0）。
86          jmp e0_e1
87  set_e1: movb $2,e0              // 收到扫描前导码 0xe1 时，设置 e1 标志（位 1）。
88          jmp e0_e1
89
90  /*
91   * This routine fills the buffer with max 8 bytes, taken from
92   * %ebx:%eax. (%edx is high). The bytes are written in the
93   * order %al,%ah,%eal,%eah,%bl,%bh ... until %eax is zero.
94   */
    /*
     * 下面该子程序把 ebx:eax 中的最多 8 个字符添入缓冲队列中。（ebx 是
     * 高字）所写入字符的顺序是 al,ah,eal,eah,bl,bh...直到 eax 等于 0。
```

```
 */
// 首先从缓冲队列地址表 table_list（tty_io.c，99 行）取控制台的读缓冲队列 read_q 地址。
// 然后把 al 寄存器中的字符复制到读队列头指针处并把头指针前移 1 字节位置。若头指针移出
// 读缓冲区的末端，就让其回绕到缓冲区开始处。 然后再看看此时缓冲队列是否已满，即比较
// 一下队列头指针是否与尾指针相等（相等表示满）。 如果已满，就把 ebx:eax 中可能还有的
// 其余字符全部抛弃掉。如果缓冲区还未满，就把 ebx:eax 中数据联合右移 8 个比特（即把 ah
// 值移到 al、bl→ah、bh→bl），然后重复上面对 al 的处理过程。直到所有字符都处理完后，
// 就保存当前头指针值，再检查一下是否有进程等待着读队列，如果有救唤醒之。
put_queue:
        pushl %ecx
        pushl %edx                      // 下句取控制台 tty 结构中读缓冲队列指针。
        movl _table_list,%edx           # read-queue for console
        movl head(%edx),%ecx            // 取队列头指针→ecx。
1:      movb %al,buf(%edx,%ecx)         // 将 al 中的字符放入头指针位置处。
        incl %ecx                       // 头指针前移 1 字节。
        andl $size-1,%ecx               // 调整头指针。若超出缓冲区末端则绕回开始处。
        cmpl tail(%edx),%ecx            # buffer full - discard everything
                                        // 头指针==尾指针吗？（即缓冲队列满了吗？）
        je 3f                           // 如果已满，则后面未放入的字符全抛弃。
        shrdl $8,%ebx,%eax              // 将 ebx 中 8 个比特右移 8 位到 eax 中，ebx 不变。
        je 2f                           // 还有字符吗？若没有（等于 0）则跳转。
        shrl $8,%ebx                    // 将 ebx 值右移 8 位，并跳转到标号 1 继续操作。
        jmp 1b
2:      movl %ecx,head(%edx)            // 若已将所有字符都放入队列，则保存头指针。
        movl proc_list(%edx),%ecx       // 该队列的等待进程指针？
        testl %ecx,%ecx                 // 检测是否有等待该队列的进程。
        je 3f                           // 无，则跳转；
        movl $0,(%ecx)                  // 有，则唤醒进程（置该进程为就绪状态）。
3:      popl %edx
        popl %ecx
        ret

// 从这里开始是键跳转表 key_table 中指针对应的各个按键（或松键）处理子程序。供上面第
// 53 行语句调用。键跳转表 key_table 在第 513 行开始。
//
// 下面这段代码根据 ctrl 或 alt 的扫描码，分别设置模式标志 mode 中相应位。如果在该扫描
// 码之前收到过 0xe0 扫描码（e0 标志置位），则说明按下的是键盘右边的 ctrl 或 alt 键，则
// 对应设置 ctrl 或 alt 在模式标志 mode 中的比特位。
ctrl:   movb $0x04,%al                  // 0x4 是 mode 中左 ctrl 键对应的比特位（位 2）。
        jmp 1f
alt:    movb $0x10,%al                  // 0x10 是 mode 中左 alt 键对应的比特位（位 4）。
1:      cmpb $0,e0                      // e0 置位了吗（按下的是右边的 ctrl/alt 键吗）？
        je 2f                           // 不是则转。
        addb %al,%al                    // 是，则改成置相应右键标志位（位 3 或位 5）。
2:      orb %al,mode                    // 设置 mode 标志中对应的比特位。
        ret
// 这段代码处理 ctrl 或 alt 键松开时的扫描码，复位模式标志 mode 中的对应比特位。在处理
// 时要根据 e0 标志是否置位来判断是否是键盘右边的 ctrl 或 alt 键。
unctrl: movb $0x04,%al                  // mode 中左 ctrl 键对应的比特位（位 2）。
        jmp 1f
unalt:  movb $0x10,%al                  // 0x10 是 mode 中左 alt 键对应的比特位（位 4）。
1:      cmpb $0,e0                      // e0 置位了吗（释放的是右边的 ctrl/alt 键吗）？
        je 2f                           // 不是，则转。
```

```
131         addb %al,%al                    // 是，则改成复位相应右键的标志位（位 3 或位 5）。
132 2:      notb %al                        // 复位 mode 标志中对应的比特位。
133         andb %al,mode
134         ret
135
    // 这段代码处理左、右 shift 键按下和松开时的扫描码，分别设置和复位 mode 中的相应位。
136 lshift:
137         orb $0x01,mode                  // 是左 shift 键按下，设置 mode 中位 0。
138         ret
139 unlshift:
140         andb $0xfe,mode                 // 是左 shift 键松开，复位 mode 中位 0。
141         ret
142 rshift:
143         orb $0x02,mode                  // 是右 shift 键按下，设置 mode 中位 1。
144         ret
145 unrshift:
146         andb $0xfd,mode                 // 是右 shift 键松开，复位 mode 中位 1。
147         ret
148
    // 这段代码对收到 caps 键扫描码进行处理。通过 mode 中位 7 可以知道 caps 键当前是否正处于
    // 在按下状态。若是则返回，否则就翻转 mode 标志中 caps 键按下的比特位（位 6）和 leds 标
    // 志中 caps-lock 比特位（位 2），设置 mode 标志中 caps 键已按下标志位（位 7）。
149 caps:   testb $0x80,mode                // 测试 mode 中位 7 是否已置位（即在按下状态）。
150         jne 1f                          // 如果已处于按下状态，则返回（186 行）。
151         xorb $4,leds                    // 翻转 leds 标志中 caps-lock 比特位（位 2）。
152         xorb $0x40,mode                 // 翻转 mode 标志中 caps 键按下的比特位（位 6）。
153         orb $0x80,mode                  // 设置 mode 标志中 caps 键已按下标志位（位 7）。
    // 这段代码根据 leds 标志，开启或关闭 LED 指示器。
154 set_leds:
155         call kb_wait                    // 等待键盘控制器输入缓冲空。
156         movb $0xed,%al                  /* set leds command */
157         outb %al,$0x60                  // 发送键盘命令 0xed 到 0x60 端口。
158         call kb_wait
159         movb leds,%al                   // 取 leds 标志，作为参数。
160         outb %al,$0x60                  // 发送该参数。
161         ret
162 uncaps: andb $0x7f,mode                 // caps 键松开，则复位 mode 中的对应位（位 7）。
163         ret
164 scroll:
165         testb $0x03,mode                // 若此时 ctrl 键也同时按下，则
166         je 1f
167         call _show_mem                  // 显示内存状态信息（mm/memory.c，457 行）。
168         jmp 2f
169 1:      call _show_state                // 否则显示进程状态信息（kernel/sched.c，45 行）。
170 2:      xorb $1,leds                    // scroll 键按下，则翻转 leds 中对应位（位 0）。
171         jmp set_leds                    // 根据 leds 标志重新开启或关闭 LED 指示器。
172 num:    xorb $2,leds                    // num 键按下，则翻转 leds 中的对应位（位 1）。
173         jmp set_leds                    // 根据 leds 标志重新开启或关闭 LED 指示器。
174
175 /*
176  *  curosr-key/numeric keypad cursor keys are handled here.
177  *  checking for numeric keypad etc.
178  */
```

```
    /*
     * 这里处理方向键/数字小键盘方向键，检测数字小键盘等。
     */
179 cursor:
180         subb $0x47,%al          // 扫描码是数字键盘上的键（其扫描码>=0x47）发出的？
181         jb 1f                   // 如果小于则不处理，返回（198 行）。
182         cmpb $12,%al            // 如果扫描码 > 0x53（0x53 - 0x47 = 12），则
183         ja 1f                   // 表示扫描码值超过 83（0x53），不处理，返回。
184         jne cur2                /* check for ctrl-alt-del */ /* 检测 ctrl-alt-del 键*/
    // 若等于 12，说明 del 键已被按下，则继续判断 ctrl 和 alt 是否也被同时按下。
185         testb $0x0c,mode        // 有 ctrl 键按下了吗？无，则跳转。
186         je cur2
187         testb $0x30,mode        // 有 alt 键按下吗？
188         jne reboot              // 有，则跳转到重启动处理（594 行）。
189 cur2:   cmpb $0x01,e0           /* e0 forces cursor movement */ /* e0 置位指光标移动*/
                                    // e0 标志置位了吗？
190         je cur                  // 置位了，则跳转光标移动处理处 cur。
191         testb $0x02,leds        /* not num-lock forces cursor */ /* num-lock 键则不许*/
                                    // 测试 leds 中标志 num-lock 键标志是否置位。
192         je cur                  // 若没有置位（num 的 LED 不亮），则也处理光标移动。
193         testb $0x03,mode        /* shift forces cursor */ /* shift 键也使光标移动 */
                                    // 测试模式标志 mode 中 shift 按下标志。
194         jne cur                 // 如果有 shift 键按下，则也进行光标移动处理。
195         xorl %ebx,%ebx          // 否则查询小数字表（199 行），取键的数字 ASCII 码。
196         movb num_table(%eax),%al        // 以 eax 作为索引值，取对应数字字符➔al。
197         jmp put_queue           // 字符放入缓冲队列中。由于要放入队列的字符数<=4，因此
198 1:      ret                     // 在执行 put_queue 前需把 ebx 清零，见 87 行上的注释。
199
    // 这段代码处理光标移动或插入删除按键。
200 cur:    movb cur_table(%eax),%al        // 取光标字符表中相应键的代表字符➔al。
201         cmpb $'9,%al            // 若字符<='9'（5、6、2 或 3），说明是上一页、下一页、
202         ja ok_cur               // 插入或删除键，则功能字符序列中要添入字符'~'。不过
203         movb $'~,%ah            // 本内核并没有对它们进行识别和处理。
204 ok_cur: shll $16,%eax          // 将 ax 中内容移到 eax 高字中。
205         movw $0x5b1b,%ax        // 把'esc ['放入 ax，与 eax 高字中字符组成移动序列。
206         xorl %ebx,%ebx          // 由于只需把 eax 中字符放入队列，因此需要把 ebx 清零。
207         jmp put_queue           // 将该字符放入缓冲队列中。
208
209 #if defined(KBD_FR)
210 num_table:
211         .ascii "789 456 1230."  // 数字小键盘上键对应的数字 ASCII 码表。
212 #else
213 num_table:
214         .ascii "789 456 1230,"
215 #endif
216 cur_table:
217         .ascii "HA5 DGC YB623"  // 小键盘上方向键或插入删除键对应的移动表示字符表。
218
219 /*
220  * this routine handles function keys
221  */
    /*
     * 下面子程序处理功能键。
```

```
 */
// 把功能键扫描码变换成转义字符序列并存放到读队列中。
func:
        subb $0x3B,%al              // 键'F1'的扫描码是 0x3B，因此 al 中是功能键索引号。
        jb end_func                 // 如果扫描码小于 0x3b，则不处理，返回。
        cmpb $9,%al                 // 功能键是 F1--F10?
        jbe ok_func                 // 是，则跳转。
        subb $18,%al                // 是功能键 F11，F12 吗？F11、F12 扫描码是 0x57、0x58。
        cmpb $10,%al                // 是功能键 F11?
        jb end_func                 // 不是，则不处理，返回。
        cmpb $11,%al                // 是功能键 F12?
        ja end_func                 // 不是，则不处理，返回。
ok_func:
        testb $0x10,mode            // 左 alt 键也同时按下了吗?
        jne alt_func                // 是则跳转处理更换虚拟控制终端。
        cmpl $4,%ecx                /* check that there is enough room */ /*检查空间*/
        jl end_func                 // [??]需要放入 4 个字符，如果放不下，则返回。
        movl func_table(,%eax,4),%eax       // 取功能键对应字符序列。
        xorl %ebx,%ebx              // 因要放入队列字符数=4，因此执行 put_queue 之前
        jmp put_queue               // 需把 ebx 清零。
// 处理 alt + Fn 组合键，改变虚拟控制终端。此时 eax 中是功能键索引号（F1 -- 0），对应
// 虚拟控制终端号。
alt_func:
        pushl %eax                  // 虚拟控制终端号入栈，作为参数。
        call _change_console        // 更改当前虚拟控制终端（chr_dev/tty_io.c，87 行）。
        popl %eax                   // 丢弃参数。
end_func:
        ret

/*
 * function keys send F1:'esc [ [ A' F2:'esc [ [ B' etc.
 */
/*
 * 功能键发送的扫描码，F1 键为：'esc [ [ A'， F2 键为：'esc [ [ B'等。
 */
func_table:
        .long 0x415b5b1b,0x425b5b1b,0x435b5b1b,0x445b5b1b
        .long 0x455b5b1b,0x465b5b1b,0x475b5b1b,0x485b5b1b
        .long 0x495b5b1b,0x4a5b5b1b,0x4b5b5b1b,0x4c5b5b1b

// 扫描码-ASCII 字符映射表。
// 根据前面定义的键盘类型(FINNISH，US，GERMEN，FRANCH)，将相应键的扫描码映射到
// ASCII 字符。
#if     defined(KBD_FINNISH)    // 以下是芬兰语键盘的扫描码映射表。
key_map:
        .byte 0,27              // 扫描码 0x00,0x01 对应的 ASCII 码；
        .ascii "1234567890+'"   // 扫描码 0x02,...0x0c,0x0d 对应的 ASCII 码，以下类似。
        .byte 127,9
        .ascii "qwertyuiop}"
        .byte 0,13,0
        .ascii "asdfghjkl|{"
        .byte 0,0
        .ascii "'zxcvbnm,.-"
```

```
265         .byte 0,'*,0,32         /* 36-39 */   /* 扫描码 0x36-0x39 对应的 ASCII 码 */
266         .fill 16,1,0            /* 3A-49 */   /* 扫描码 0x3A-0x49 对应的 ASCII 码 */
267         .byte '-,0,0,0,'+       /* 4A-4E */   /* 扫描码 0x4A-0x4E 对应的 ASCII 码 */
268         .byte 0,0,0,0,0,0,0     /* 4F-55 */   /* 扫描码 0x4F-0x55 对应的 ASCII 码 */
269         .byte '<
270         .fill 10,1,0
271
272 shift_map:                      // shift 键同时按下时的映射表。
273         .byte 0,27
274         .ascii "!\"#$%&/()=?`"
275         .byte 127,9
276         .ascii "QWERTYUIOP]^"
277         .byte 13,0
278         .ascii "ASDFGHJKL\\["
279         .byte 0,0
280         .ascii "*ZXCVBNM;:_"
281         .byte 0,'*,0,32         /* 36-39 */
282         .fill 16,1,0            /* 3A-49 */
283         .byte '-,0,0,0,'+       /* 4A-4E */
284         .byte 0,0,0,0,0,0,0     /* 4F-55 */
285         .byte '>
286         .fill 10,1,0
287
288 alt_map:                         // alt 键同时按下时的映射表。
289         .byte 0,0
290         .ascii "\0@\0$\0\0{[]}\\\0"
291         .byte 0,0
292         .byte 0,0,0,0,0,0,0,0,0,0,0
293         .byte '~,13,0
294         .byte 0,0,0,0,0,0,0,0,0,0,0
295         .byte 0,0
296         .byte 0,0,0,0,0,0,0,0,0,0,0
297         .byte 0,0,0,0           /* 36-39 */
298         .fill 16,1,0            /* 3A-49 */
299         .byte 0,0,0,0,0         /* 4A-4E */
300         .byte 0,0,0,0,0,0,0     /* 4F-55 */
301         .byte '|
302         .fill 10,1,0
303
304 #elif defined(KBD_US)           // 以下是美式键盘的扫描码映射表。
305
306 key_map:
307         .byte 0,27
308         .ascii "1234567890-="
309         .byte 127,9
310         .ascii "qwertyuiop[]"
311         .byte 13,0
312         .ascii "asdfghjkl;'"
313         .byte '`,0
314         .ascii "\\zxcvbnm,./"
315         .byte 0,'*,0,32         /* 36-39 */
316         .fill 16,1,0            /* 3A-49 */
317         .byte '-,0,0,0,'+       /* 4A-4E */
```

```
318         .byte 0,0,0,0,0,0,0     /* 4F-55 */
319         .byte '<
320         .fill 10,1,0
321
322
323 shift_map:
324         .byte 0,27
325         .ascii "!@#$%^&*()_+"
326         .byte 127,9
327         .ascii "QWERTYUIOP{}"
328         .byte 13,0
329         .ascii "ASDFGHJKL:\""
330         .byte '~,0
331         .ascii "|ZXCVBNM<>?"
332         .byte 0,'*,0,32         /* 36-39 */
333         .fill 16,1,0            /* 3A-49 */
334         .byte '-,0,0,0,'+       /* 4A-4E */
335         .byte 0,0,0,0,0,0,0     /* 4F-55 */
336         .byte '>
337         .fill 10,1,0
338
339 alt_map:
340         .byte 0,0
341         .ascii "\0@\0$\0\0{[]}\\\0"
342         .byte 0,0
343         .byte 0,0,0,0,0,0,0,0,0,0,0
344         .byte '~,13,0
345         .byte 0,0,0,0,0,0,0,0,0,0,0
346         .byte 0,0
347         .byte 0,0,0,0,0,0,0,0,0,0,0
348         .byte 0,0,0,0           /* 36-39 */
349         .fill 16,1,0            /* 3A-49 */
350         .byte 0,0,0,0,0         /* 4A-4E */
351         .byte 0,0,0,0,0,0,0     /* 4F-55 */
352         .byte '|
353         .fill 10,1,0
354
355 #elif defined(KBD_GR)           // 以下是德语键盘的扫描码映射表。
356
357 key_map:
358         .byte 0,27
359         .ascii "1234567890\\'"
360         .byte 127,9
361         .ascii "qwertzuiop@+"
362         .byte 13,0
363         .ascii "asdfghjkl[]^"
364         .byte 0,'#
365         .ascii "yxcvbnm,.-"
366         .byte 0,'*,0,32         /* 36-39 */
367         .fill 16,1,0            /* 3A-49 */
368         .byte '-,0,0,0,'+       /* 4A-4E */
369         .byte 0,0,0,0,0,0,0     /* 4F-55 */
370         .byte '<
```

```
371         .fill 10,1,0
372
373
374 shift_map:
375         .byte 0,27
376         .ascii "!\"#$%&/()=?`"
377         .byte 127,9
378         .ascii "QWERTZUIOP\\*"
379         .byte 13,0
380         .ascii "ASDFGHJKL{}~"
381         .byte 0,''
382         .ascii "YXCVBNM;:_"
383         .byte 0,'*,0,32          /* 36-39 */
384         .fill 16,1,0             /* 3A-49 */
385         .byte '-,0,0,0,'+        /* 4A-4E */
386         .byte 0,0,0,0,0,0,0      /* 4F-55 */
387         .byte '>
388         .fill 10,1,0
389
390 alt_map:
391         .byte 0,0
392         .ascii "\0@\0$\0\0{[]}\\\0"
393         .byte 0,0
394         .byte '@,0,0,0,0,0,0,0,0,0,0
395         .byte '~,13,0
396         .byte 0,0,0,0,0,0,0,0,0,0,0
397         .byte 0,0
398         .byte 0,0,0,0,0,0,0,0,0,0,0
399         .byte 0,0,0,0            /* 36-39 */
400         .fill 16,1,0             /* 3A-49 */
401         .byte 0,0,0,0,0          /* 4A-4E */
402         .byte 0,0,0,0,0,0,0      /* 4F-55 */
403         .byte '|
404         .fill 10,1,0
405
406
407 #elif defined(KBD_FR)            // 以下是法语键盘的扫描码映射表。
408
409 key_map:
410         .byte 0,27
411         .ascii "&{\"'(-}_/@)="
412         .byte 127,9
413         .ascii "azertyuiop^$"
414         .byte 13,0
415         .ascii "qsdfghjklm|"
416         .byte '`,0,42            /* coin sup gauche, don't know, [*|mu] */
417         .ascii "wxcvbn,;:!"
418         .byte 0,'*,0,32          /* 36-39 */
419         .fill 16,1,0             /* 3A-49 */
420         .byte '-,0,0,0,'+        /* 4A-4E */
421         .byte 0,0,0,0,0,0,0      /* 4F-55 */
422         .byte '<
423         .fill 10,1,0
```

```
424
425 shift_map:
426         .byte 0,27
427         .ascii "1234567890]+"
428         .byte 127,9
429         .ascii "AZERTYUIOP<>"
430         .byte 13,0
431         .ascii "QSDFGHJKLM%"
432         .byte '~,0,'#
433         .ascii "WXCVBN?./\\"
434         .byte 0,'*,0,32         /* 36-39 */
435         .fill 16,1,0            /* 3A-49 */
436         .byte '-,0,0,0,'+       /* 4A-4E */
437         .byte 0,0,0,0,0,0,0     /* 4F-55 */
438         .byte '>
439         .fill 10,1,0
440
441 alt_map:
442         .byte 0,0
443         .ascii "\0~#{[|`\\^@]}"
444         .byte 0,0
445         .byte '@,0,0,0,0,0,0,0,0,0,0
446         .byte '~,13,0
447         .byte 0,0,0,0,0,0,0,0,0,0,0
448         .byte 0,0
449         .byte 0,0,0,0,0,0,0,0,0,0,0
450         .byte 0,0,0,0           /* 36-39 */
451         .fill 16,1,0            /* 3A-49 */
452         .byte 0,0,0,0,0         /* 4A-4E */
453         .byte 0,0,0,0,0,0,0     /* 4F-55 */
454         .byte '|
455         .fill 10,1,0
456
457 #else
458 #error "KBD-type not defined"
459 #endif
460 /*
461  * do_self handles "normal" keys, ie keys that don't change meaning
462  * and which have just one character returns.
463  */
    /*
     * do_self 用于处理“普通”键，也即含义没有变化并且只有一个字符返回的键。
     */
    // 首先根据 mode 标志选择 alt_map、shift_map 或 key_map 映射表之一。
464 do_self:
465         lea alt_map,%ebx            // 取 alt 键同时按下时的映射表基址 alt_map。
466         testb $0x20,mode            /* alt-gr */  /* 右 alt 键同时按下了？ */
467         jne 1f                      // 是，则向前跳转到标号 1 处。
468         lea shift_map,%ebx          // 取 shift 键同时按下时的映射表基址 shift_map。
469         testb $0x03,mode            // 有 shift 键同时按下了吗？
470         jne 1f                      // 有，则向前跳转到标号 1 处去映射字符。
471         lea key_map,%ebx            // 否则使用普通映射表 key_map。
    // 然后根据扫描码取映射表中对应的 ASCII 字符。若没有对应字符，则返回（转 none）。
```

```
1:      movb (%ebx,%eax),%al            // 将扫描码作为索引值，取对应的 ASCII 码➔al。
        orb %al,%al                     // 检测看是否有对应的 ASCII 码。
        je none                         // 若没有(对应的 ASCII 码=0)，则返回。
// 若 ctrl 键已按下或 caps 键锁定，并且字符在 'a'--'}'（0x61--0x7D）范围内，则将其转成
// 大写字符（0x41--0x5D）。
        testb $0x4c,mode                /* ctrl or caps */  /* 控制键已按下或 caps 亮? */
        je 2f                           // 没有，则向前跳转标号 2 处。
        cmpb $'a,%al                    // 将 al 中的字符与'a'比较。
        jb 2f                           // 若 al 值<'a'，则转标号 2 处。
        cmpb $'},%al                    // 将 al 中的字符与'}'比较。
        ja 2f                           // 若 al 值>'}'，则转标号 2 处。
        subb $32,%al                    // 将 al 转换为大写字符（减 0x20）。
// 若 ctrl 键已按下，并且字符在 ``--'_'（0x40--0x5F）之间，即是大写字符，则将其转换为
// 控制字符（0x00--0x1F）。
2:      testb $0x0c,mode                /* ctrl */  /* ctrl 键同时按下了吗? */
        je 3f                           // 若没有则转标号 3。
        cmpb $64,%al                    // 将 al 与'@'（64）字符比较，即判断字符所属范围。
        jb 3f                           // 若值<'@'，则转标号 3。
        cmpb $64+32,%al                 // 将 al 与``（96）字符比较，即判断字符所属范围。
        jae 3f                          // 若值>=``，则转标号 3。
        subb $64,%al                    // 否则 al 减 0x40，转换为 0x00--0x1f 的控制字符。
// 若左 alt 键同时按下，则将字符的位 7 置位。即此时生成值大于 0x7f 的扩展字符集中的字符。
3:      testb $0x10,mode                /* left alt */  /* 左 alt 键同时按下? */
        je 4f                           // 没有，则转标号 4。
        orb $0x80,%al                   // 字符的位 7 置位。
// 将 al 中的字符放入读缓冲队列中。
4:      andl $0xff,%eax                 // 清 eax 的高字和 ah。
        xorl %ebx,%ebx                  // 由于放入队列字符数<=4，因此需把 ebx 清零。
        call put_queue                  // 将字符放入缓冲队列中。
none:   ret

/*
 * minus has a routine of it's own, as a 'E0h' before
 * the scan code for minus means that the numeric keypad
 * slash was pushed.
 */
/*
 * 减号有它自己的处理子程序，因为在减号扫描码之前的 0xe0
 * 意味着按下了数字小键盘上的斜杠键。
 */
// 注意，对于芬兰语和德语键盘，扫描码 0x35 对应的是'-'键。参见第 264 和 365 行。
minus:  cmpb $1,e0                      // e0 标志置位了吗?
        jne do_self                     // 没有，则调用 do_self 对减号符进行普通处理。
        movl $'/,%eax                   // 否则用'/'替换减号'-'➔al。
        xorl %ebx,%ebx                  // 由于放入队列字符数<=4，因此需把 ebx 清零。
        jmp put_queue                   // 并将字符放入缓冲队列中。

/*
 * This table decides which routine to call when a scan-code has been
 * gotten. Most routines just call do_self, or none, depending if
 * they are make or break.
 */
/*
```

```
    * 下面是一张子程序地址跳转表。当取得扫描码后就根据此表调用相应的扫描码
    * 处理子程序。大多数调用的子程序是 do_self，或者是 none，这起决于是按键
    * （make）还是释放键(break)。
    */
key_table:
        .long none,do_self,do_self,do_self      /* 00-03 s0 esc 1 2 */
        .long do_self,do_self,do_self,do_self   /* 04-07 3 4 5 6 */
        .long do_self,do_self,do_self,do_self   /* 08-0B 7 8 9 0 */
        .long do_self,do_self,do_self,do_self   /* 0C-0F + ' bs tab */
        .long do_self,do_self,do_self,do_self   /* 10-13 q w e r */
        .long do_self,do_self,do_self,do_self   /* 14-17 t y u i */
        .long do_self,do_self,do_self,do_self   /* 18-1B o p } ^ */
        .long do_self,ctrl,do_self,do_self      /* 1C-1F enter ctrl a s */
        .long do_self,do_self,do_self,do_self   /* 20-23 d f g h */
        .long do_self,do_self,do_self,do_self   /* 24-27 j k l | */
        .long do_self,do_self,lshift,do_self    /* 28-2B { para lshift , */
        .long do_self,do_self,do_self,do_self   /* 2C-2F z x c v */
        .long do_self,do_self,do_self,do_self   /* 30-33 b n m , */
        .long do_self,minus,rshift,do_self      /* 34-37 . - rshift * */
        .long alt,do_self,caps,func             /* 38-3B alt sp caps f1 */
        .long func,func,func,func               /* 3C-3F f2 f3 f4 f5 */
        .long func,func,func,func               /* 40-43 f6 f7 f8 f9 */
        .long func,num,scroll,cursor            /* 44-47 f10 num scr home */
        .long cursor,cursor,do_self,cursor      /* 48-4B up pgup - left */
        .long cursor,cursor,do_self,cursor      /* 4C-4F n5 right + end */
        .long cursor,cursor,cursor,cursor       /* 50-53 dn pgdn ins del */
        .long none,none,do_self,func            /* 54-57 sysreq ? < f11 */
        .long func,none,none,none               /* 58-5B f12 ? ? ? */
        .long none,none,none,none               /* 5C-5F ? ? ? ? */
        .long none,none,none,none               /* 60-63 ? ? ? ? */
        .long none,none,none,none               /* 64-67 ? ? ? ? */
        .long none,none,none,none               /* 68-6B ? ? ? ? */
        .long none,none,none,none               /* 6C-6F ? ? ? ? */
        .long none,none,none,none               /* 70-73 ? ? ? ? */
        .long none,none,none,none               /* 74-77 ? ? ? ? */
        .long none,none,none,none               /* 78-7B ? ? ? ? */
        .long none,none,none,none               /* 7C-7F ? ? ? ? */
        .long none,none,none,none               /* 80-83 ? br br br */
        .long none,none,none,none               /* 84-87 br br br br */
        .long none,none,none,none               /* 88-8B br br br br */
        .long none,none,none,none               /* 8C-8F br br br br */
        .long none,none,none,none               /* 90-93 br br br br */
        .long none,none,none,none               /* 94-97 br br br br */
        .long none,none,none,none               /* 98-9B br br br br */
        .long none,unctrl,none,none             /* 9C-9F br unctrl br br */
        .long none,none,none,none               /* A0-A3 br br br br */
        .long none,none,none,none               /* A4-A7 br br br br */
        .long none,none,unlshift,none           /* A8-AB br br unlshift br */
        .long none,none,none,none               /* AC-AF br br br br */
        .long none,none,none,none               /* B0-B3 br br br br */
        .long none,none,unrshift,none           /* B4-B7 br br unrshift br */
        .long unalt,none,uncaps,none            /* B8-BB unalt br uncaps br */
        .long none,none,none,none               /* BC-BF br br br br */
```

```
562         .long none,none,none,none               /* C0-C3 br br br br */
563         .long none,none,none,none               /* C4-C7 br br br br */
564         .long none,none,none,none               /* C8-CB br br br br */
565         .long none,none,none,none               /* CC-CF br br br br */
566         .long none,none,none,none               /* D0-D3 br br br br */
567         .long none,none,none,none               /* D4-D7 br br br br */
568         .long none,none,none,none               /* D8-DB br ? ? ? */
569         .long none,none,none,none               /* DC-DF ? ? ? ? */
570         .long none,none,none,none               /* E0-E3 e0 e1 ? ? */
571         .long none,none,none,none               /* E4-E7 ? ? ? ? */
572         .long none,none,none,none               /* E8-EB ? ? ? ? */
573         .long none,none,none,none               /* EC-EF ? ? ? ? */
574         .long none,none,none,none               /* F0-F3 ? ? ? ? */
575         .long none,none,none,none               /* F4-F7 ? ? ? ? */
576         .long none,none,none,none               /* F8-FB ? ? ? ? */
577         .long none,none,none,none               /* FC-FF ? ? ? ? */
578
579 /*
580  * kb_wait waits for the keyboard controller buffer to empty.
581  * there is no timeout - if the buffer doesn't empty, we hang.
582  */
    /*
     * 子程序 kb_wait 用于等待键盘控制器缓冲空。不存在超时处理 - 如果
     * 缓冲永远不空的话，程序就会永远等待(死掉)。
     */
583 kb_wait:
584         pushl %eax
585 1:      inb $0x64,%al                   // 读键盘控制器状态。
586         testb $0x02,%al                 // 测试输入缓冲器是否为空（等于 0）。
587         jne 1b                          // 若不空，则跳转循环等待。
588         popl %eax
589         ret
590 /*
591  * This routine reboots the machine by asking the keyboard
592  * controller to pulse the reset-line low.
593  */
    /*
     * 该子程序通过设置键盘控制器，向复位线输出负脉冲，使系统复
     * 位重启（reboot）。
     */
    // 该子程序往物理内存地址 0x472 处写值 0x1234。该位置是启动模式（reboot_mode）标志字。
    // 在启动过程中 ROM BIOS 会读取该启动模式标志值并根据其值来指导下一步的执行。 如果该
    // 值是 0x1234，则 BIOS 就会跳过内存检测过程而执行热启动（Warm-boot）过程。 如果若该
    // 值为 0，则执行冷启动（Cold-boot）过程。
594 reboot:
595         call kb_wait                    // 首先等待键盘控制器输入缓冲器空。
596         movw $0x1234,0x472              /* don't do memory check */ /* 不检测内存 */
597         movb $0xfc,%al                  /* pulse reset and A20 low */
598         outb %al,$0x64                  // 向系统复位引脚和 A20 线输出负脉冲。
599 die:    jmp die                         // 停机。
```