程序 10-3 linux/kernel/chr_drv/serial.c

```
1  /*
2   *  linux/kernel/serial.c
3   *
4   *  (C) 1991  Linus Torvalds
5   */
6
7  /*
8   *      serial.c
9   *
10  * This module implements the rs232 io functions
11  *      void rs_write(struct tty_struct * queue);
12  *      void rs_init(void);
13  * and all interrupts pertaining to serial IO.
14  */
    /*
     *     serial.c
     * 该程序用于实现 rs232 的输入输出函数
     *     void rs_write(struct tty_struct *queue);
     *     void rs_init(void);
     * 以及与串行 IO 有关系的所有中断处理程序。
     */
15
16 #include <linux/tty.h>    // tty 头文件，定义了有关 tty_io，串行通信方面的参数、常数。
17 #include <linux/sched.h>  // 调度程序头文件，定义了任务结构 task_struct、任务 0 数据等。
18 #include <asm/system.h>   // 系统头文件。定义设置或修改描述符/中断门等的嵌入式汇编宏。
19 #include <asm/io.h>       // io 头文件。定义硬件端口输入/输出宏汇编语句。
20
21 #define WAKEUP_CHARS (TTY_BUF_SIZE/4)  // 当写队列中含有 WAKEUP_CHARS 个字符时就开始发送。
22
23 extern void rs1_interrupt(void);       // 串行口 1 的中断处理程序（rs_io.s, 34 行）。
24 extern void rs2_interrupt(void);       // 串行口 2 的中断处理程序（rs_io.s, 38 行）。
25
   //// 初始化串行端口
   // 设置指定串行端口的传输波特率（2400bps）并允许除了写保持寄存器空以外的所有中断源。
   // 另外，在输出 2 字节的波特率因子时，须首先设置线路控制寄存器的 DLAB 位（位 7）。
   // 参数：port 是串行端口基地址，串口 1 - 0x3F8；串口 2 - 0x2F8。
26 static void init(int port)
27 {
28         outb_p(0x80,port+3);    /* set DLAB of line control reg */
29         outb_p(0x30,port);      /* LS of divisor (48 -> 2400 bps */
30         outb_p(0x00,port+1);    /* MS of divisor */
31         outb_p(0x03,port+3);    /* reset DLAB */
32         outb_p(0x0b,port+4);    /* set DTR,RTS, OUT_2 */
33         outb_p(0x0d,port+1);    /* enable all intrs but writes */
34         (void)inb(port);        /* read data port to reset things (?) */
35 }
36
   //// 初始化串行中断程序和串行接口。
   // 中断描述符表 IDT 中的门描述符设置宏 set_intr_gate()在 include/asm/system.h 中实现。
37 void rs_init(void)
38 {
   // 下面两句用于设置两个串行口的中断门描述符。rs1_interrupt 是串口 1 的中断处理过程指针。
```

```
   // 串口 1 使用的中断是 int 0x24，串口 2 的是 int 0x23。参见表 2-2 和 system.h 文件。
39         set_intr_gate(0x24,rs1_interrupt);    // 设置串行口 1 的中断门向量(IRQ4 信号)。
40         set_intr_gate(0x23,rs2_interrupt);    // 设置串行口 2 的中断门向量(IRQ3 信号)。
41         init(tty_table[64].read_q->data);     // 初始化串行口 1(.data 是端口基地址)。
42         init(tty_table[65].read_q->data);     // 初始化串行口 2。
43         outb(inb_p(0x21)&0xE7,0x21);          // 允许主 8259A 响应 IRQ3、IRQ4 中断请求。
44 }
45
46 /*
47  * This routine gets called when tty_write has put something into
48  * the write_queue. It must check wheter the queue is empty, and
49  * set the interrupt register accordingly
50  *
51  *      void _rs_write(struct tty_struct * tty);
52  */
   /*
    * 在 tty_write()已将数据放入输出(写)队列时会调用下面的子程序。在该
    * 子程序中必须首先检查写队列是否为空，然后设置相应中断寄存器。
    */
   //// 串行数据发送输出。
   // 该函数实际上只是开启发送保持寄存器已空中断标志。此后当发送保持寄存器空时，UART 就会
   // 产生中断请求。而在该串行中断处理过程中，程序会取出写队列尾指针处的字符，并输出到发
   // 送保持寄存器中。一旦 UART 把该字符发送了出去，发送保持寄存器又会变空而引发中断请求。
   // 于是只要写队列中还有字符，系统就会重复这个处理过程，把字符一个一个地发送出去。当写
   // 队列中所有字符都发送了出去，写队列变空了，中断处理程序就会把中断允许寄存器中的发送
   // 保持寄存器中断允许标志复位掉，从而再次禁止发送保持寄存器空引发中断请求。此次“循环”
   // 发送操作也随之结束。
53 void rs_write(struct tty_struct * tty)
54 {
   // 如果写队列不空，则首先从 0x3f9（或 0x2f9）读取中断允许寄存器内容，添上发送保持寄存器
   // 中断允许标志（位 1）后，再写回该寄存器。这样，当发送保持寄存器空时 UART 就能够因期望
   // 获得欲发送的字符而引发中断。write_q.data 中是串行端口基地址。
55         cli();
56         if (!EMPTY(tty->write_q))
57                 outb(inb_p(tty->write_q->data+1)|0x02,tty->write_q->data+1);
58         sti();
59 }
60
```