RS_IO.S 注释

Coolday

1、本文件主要涉及了 RS-232 串行通信标准，RS_232C 是目前普遍使用的串行通信标准，这里只对其中使用的异步通信控制器 8250 中的寄存器做简单介绍。

一般计算机都配备两个 RS-232 接口，为其保留的地址为 0x3f8(对应 COM1),0x2f8(对应 COM2)。而 8250 控制器中包含有 10 个内部寄存器，但内部寻址线只有三根 A0,A1,A2，故有两个口地址分别由 2 个寄存器共用，当访问这两对寄存器时，由通讯线路控制寄存器的最高位 DLAB 来识别，通过 DLAB 和 A0，A1,A2 来访问这 10 个寄存器

DLAB	A2 A1 A0	寄 存 器	端口地址
0	0 0 0	接收/发送数据寄存器（RBR/THR）	3F8/2F8
0	0 0 1	中断允许寄存器(IER)	3F9/2F9
X	0 1 0	中断标识寄存器(IIR)	3FA/2FA
X	0 1 1	通讯线路控制寄存器(LCR)	3FB/2FB
X	1 0 0	MODEM 控制寄存器(MCR)	3FC/2FC
X	1 0 1	通讯线路状态寄存器(LSR)	3FD/2FD
X	1 1 0	MODEM 状态寄存器(MSR)	3FE/2FE
X	1 1 1	专用	3FF/2FF
1	0 0 0	除数锁存器（低字节）	3F8/2F8
1	0 0 1	除数锁存器（高字节）	3F9/2F9

表一：内部寄存器的选择

下面对 rs_io.s 涉及的几个寄存器格式做一解释

1） 通信线路状态寄存器 LSR(3FD/2FD)

7	6	5	4	3	2	1	0
0	TERE	THRE	BI	FE	PE	OE	DR
	发送移位寄存器空	发送保持寄存器空	间断错误	格式（帧）错	奇偶错	溢出	接收数据就绪

2） 中断允许寄存器 IER(3F9/2F9)

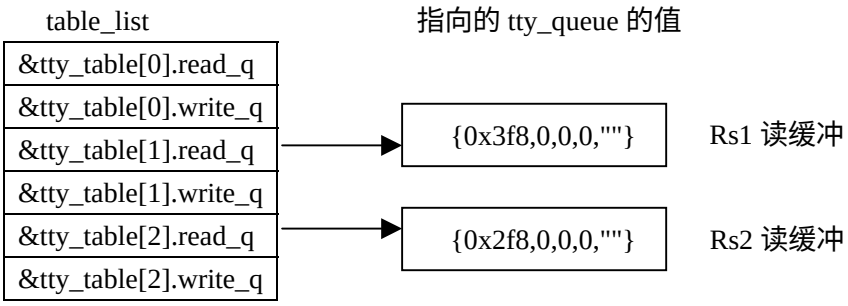
7	6	5	4	3	2	1	0
0	0	0	0	EDSSI	ELSI	ETBE	ERBFI
				1=MODEM 状态变化中断	1=允许接收有错或间接状态中断	1=允许发送保持寄存器中断	1=允许接收器数据就绪中断

3）中断识别寄存器 IIR(3FA/2FA)

7	6	5	4	3	2	1	0
0	0	0	0	0	ID1	ID0	IP
					00=MODEM 状态发生变化 01=发送保持寄存器空 10=接收数据就绪 11=接收有错或间断状态		0=有中断 1=无中断

2、相关的数据结构简介

rs_io.s 与 tty_io.c 紧密相关，其中涉及的数据结构有 table_list，tty_table，tty_queue 等其三者的关系如下图所示:



这里只画出了读缓冲，写缓冲类似，图中的值为初始值，tty_queue 结构如下：

```
 struct tty_queue {
      unsigned long data;    //端口地址
      unsigned long head;    //数据头指针
      unsigned long tail;     //数据尾指针
      struct task_struct * proc_list;    //等待进程队列
      char buf[TTY_BUF_SIZE];    //数据队列
}
```

3、rs_io.s 注释

```
/*
 *    rs_io.s
 *本模块完成 rs232    i/o 中断处理
 * This module implements the rs232 io interrupts.
 */

.text
.globl _rs1_interrupt,_rs2_interrupt

size  = 1024                    /* must be power of two !
                                and must match the value
                                in tty_io.c!!! */

/* these are the offsets into the read/write buffer structures */
//下面对应于 tty_queue 结构中各项的偏移量,参见 tty.h
rs_addr = 0
head = 4
tail = 8
proc_list = 12
buf = 16
```

```
startup     = 256           /* chars left in write queue when we restart it */

/*
 * These are the actual interrupt routines. They look where
 * the interrupt is coming from, and take appropriate action.
 */
.align 2
_rs1_interrupt:
/*  如为rs1中断，将rs1对应队列的地址(&tty_table[1].read_q)压栈,参见tty_io.c的table_list[]
*/
        pushl $_table_list+8
        jmp rs_int
.align 2
_rs2_interrupt:
/*  如为rs3中断,将rs2对应队列的地址(&tty_table[2].read_q)压栈 ,参见tty_io.c的table_list[]
*/
        pushl $_table_list+16
rs_int:
        pushl %edx
        pushl %ecx
        pushl %ebx
        pushl %eax
        push %es
        push %ds        /* as this is an interrupt, we cannot */
/*保存当前寄存器值并转入核心段  */
        pushl $0x10             /* know that bs is ok. Load it */
        pop %ds
        pushl $0x10
        pop %es
        movl 24(%esp),%edx        //edx = &tty_table[X].read_q ,X=1 或 2
        movl (%edx),%edx
        movl rs_addr(%edx),%edx    //获得 rsX 的端口地址值,即 0x3F8/0x2F8
        addl $2,%edx          /* interrupt ident. reg .得到中断识别寄存器IIR端口地址,0x3FA/2FA*/
rep_int:
        xorl %eax,%eax
        inb %dx,%al        //读入 IIR 值
        testb $1,%al       //测试有无中断，如没有则结束
        jne end
        cmpb $6,%al             /* this shouldn't happen, but ... */
        ja end                  //如果接收有错或间断则结束
        movl 24(%esp),%ecx      //ecx = &tty_table[X].read_q ,X=1 或 2
        pushl %edx
        subl $2,%edx            //edx  恢复为 0x3F8/2F8
```

```
/*调用相应的中断处理过程，根据 eax 的值分别对应：
                eax                对应函数地址
                  0                  modem_status        (jmp_table+0*2,即 modem_status)
                  2                  write_char          (jmp_table+2*2,即 write_status)
                  4                 read_char            (jmp_table+4*2,即 read_char)
                  6                 line_status          (jmp_table+6*2,即 line_status )
*/
      call jmp_table(,%eax,2)        /* NOTE! not *4, bit0 is 0 already */
      popl %edx
      jmp rep_int       //重新开始循环
end: movb $0x20,%al           //设中断服务结束标志，并输出
      outb %al,$0x20           /* EOI */
/*弹出压入的寄存器值，并将压入的 table_list 项舍弃，返回*/
      pop %ds
      pop %es
      popl %eax
      popl %ebx
      popl %ecx
      popl %edx
      addl $4,%esp         # jump over _table_list entry
      iret

jmp_table:
      .long modem_status,write_char,read_char,line_status

.align 2
modem_status:
/*  如标志为 MODEM 状态变化，则 edx =0x3FC/2FC，即 MODEM 状态寄存器 MSR
      addl $6,%edx        /* clear intr by reading modem status reg */
      inb %dx,%al          /*读入 MSR 值，不作其他操作*/
      ret

.align 2
line_status:
/*  如标志为接收出错，则 edx =0x3FD/2FD，即通信线路状态寄存器 LSR
      addl $5,%edx        /* clear intr by reading line status reg. */
      inb %dx,%al       /*读入 LSR 值，不做其他操作  */
      ret

.align 2
read_char:
/*  如为读模式，则直接读取数据接收缓冲寄存器值(3F8/2F8) */
      inb %dx,%al
```

```
        movl %ecx,%edx
        subl $_table_list,%edx      //得到&tty_table[X].read_q 的偏移量（8 或 16）
        shrl $3,%edx            //得到对应于 tty_table 的偏移量
        movl (%ecx),%ecx        //读取缓冲 read_q 地址
        movl head(%ecx),%ebx    //得到数据缓冲区的头指针
        movb %al,buf(%ecx,%ebx)    //读入的数据放入头指针对应地址
        incl %ebx
        andl $size-1,%ebx       //如果 ebx>=size-1,则重新回到 0，循环队列
/*测试头指针是否已等于尾指针，如是则返回 */
        cmpl tail(%ecx),%ebx
        je 1f
        movl %ebx,head(%ecx)    //否则修改头指针
        pushl %edx          //压入函数参数，对于 tty_table 的偏移量
        call _do_tty_interrupt    //调用 tty 处理
        addl $4,%esp      //丢弃压入的 edx
1:      ret

.align 2
write_char:
/*  如为写模式，则将指针对应写队列&tty_table[X].write_q */
        movl 4(%ecx),%ecx               # write-queue
        movl head(%ecx),%ebx
        subl tail(%ecx),%ebx
/*测试写缓冲剩余的字符数？ */
        andl $size-1,%ebx           # nr chars in queue
        je write_buffer_empty    //如为 0，即写缓冲区是否已为空，则调用 write_buff_empty 处理
        cmpl $startup,%ebx
        ja 1f         //测试剩余字符数是否超过 256，超过则跳转，否则唤醒睡眠进程
        movl proc_list(%ecx),%ebx   # wake up sleeping process，
        testl %ebx,%ebx                 # is there any?
        je 1f           //如果没有其他进程则将该队列的 pro_list 置为 0
        movl $0,(%ebx)
1:      movl tail(%ecx),%ebx
        movb buf(%ecx,%ebx),%al
        outb %al,%dx      //将尾指针指向数据输出
        incl %ebx
        andl $size-1,%ebx       //与 1023 相与，如尾指针已到缓冲区结束，则转到 0，循环队列
        movl %ebx,tail(%ecx) //修改尾指针
/*测试头指针=尾指针？,相等则缓冲区已空，调 write_buff_empty 处理 */
        cmpl head(%ecx),%ebx
        je write_buffer_empty
        ret
.align 2
write_buffer_empty:
```

```
/*唤醒写缓冲的睡眠进程*/
        movl proc_list(%ecx),%ebx   # wake up sleeping process
        testl %ebx,%ebx             # is there any?
        je 1f    //如果已没有等待进程，则将该队列的 pro_list 置为 0
        movl $0,(%ebx)
1:      incl %edx      //edx = 0x3F9/2F9,指向中断允许寄存器 IER
        inb %dx,%al    //读入 IER 值
        jmp 1f
1:      jmp 1f
/*将允许发送保持寄存器中断位（ETBE）设为 0，不允许发送中断*/
1:      andb $0xd,%al       /* disable transmit interrupt */
        outb %al,%dx
        ret
```