Práctica 8: La gestión del RS-232 se traspasa al ratón

En las prácticas previas el dipositivo físico asociado a la tarea *mouse.c* ha sido el reloj. Este dispositivo ha servido bien al propósito de provocar interrupciones y un entero asociado a cada interrupción que finalmente había de ser transportado al proceso de usuario *raton.c* a través de los buffers de bajo y alto nivel. En el camino hemos aprendido a diseñar un driver de forma estructurada y progresiva.

En esta práctica trabajaremos con un dispositivo real, un puerto RS-232, al que vamos a conectar un ratón serie. De lo que se trata es de modificar *mouse.c* para que del buffer de bajo nivel extraiga los datos que envía el ratón en lugar de los enteros sintéticos que generaba la rutina de interrupción del reloj en la práctica 7. Los datos que finalmente se elevan al programa de usuario serán objeto de la práctica 9.

1. La rutina de interrupción en MINIX.

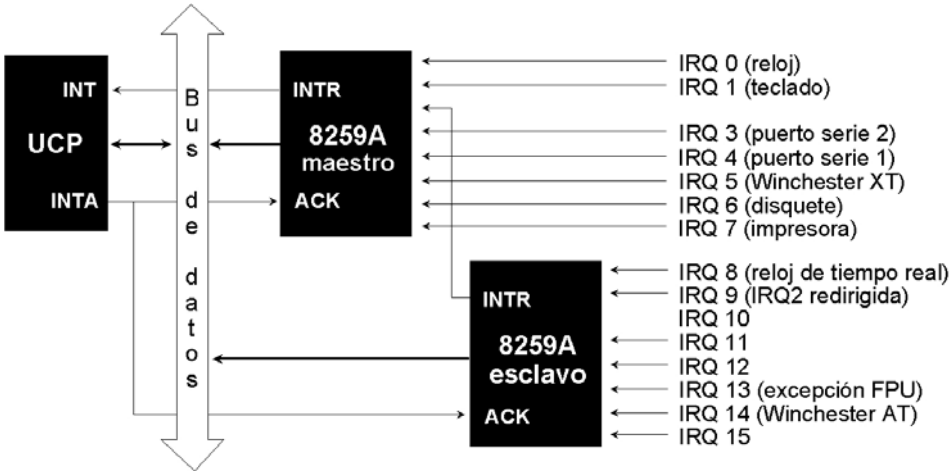

Figura 1. Las fuentes de interrupción en la arquitectura PC

Observemos la figura 1. La arquitectura PC soporta dos tarjetas RS-232, más conocidas como los "puerto serie" uno y dos, a los que asigna las interrupciones 3 y 4. El fichero */usr/src/kernel/const.h* define estas constantes:

```
#define SECONDARY_IRQ  3  /* RS232 interrupt vector for port 2 */
#define RS232_IRQ      4  /* RS232 interrupt vector for port 1 */
```

En el tema de teoría "Interrupciones" estudiamos las rutinas de interrupción. Vimos que todas respondían al mismo patrón, a saber, una macro en ensamblador a la que se pasa el parámetro 0, 1, 2, etc. según la figura 1. El código que sigue muestra esta macro.

```
#define hwint_master(irq) \
        call    save                    /* save interrupted process state */;\
        inb     INT_CTLMASK                                                 ;\
        orb     al, *[1<<irq]                                               ;\
        outb    INT_CTLMASK             /* disable the irq            */    ;\
        movb    al, *ENABLE                                                 ;\
        outb    INT_CTL                 /* reenable master 8259       */    ;\
        sti                             /* enable interrupts          */    ;\
        mov     ax, *irq                                                    ;\
        push    ax                      /* irq                        */    ;\
        call    @_irq_table + 2*irq     /* ax = (*irq_table[irq])(irq)*/    ;\
        pop     cx                                                          ;\
        cli                             /* disable interrupts         */    ;\
        test    ax, ax                  /* need to reenable irq?      */    ;\
        jz      0f                                                          ;\
        inb     INT_CTLMASK                                                 ;\
        andb    al, *~[1<<irq]                                              ;\
        outb    INT_CTLMASK             /* enable the irq             */    ;\
0:      ret                             /* restart (another) process  */
```

Aparte de salvar y restaurar el contexto, el trabajo "útil" de la macro es invocar su "función cuerpo", una función escrita en C cuya dirección de comienzo almacena el vector `irq_table`. El parámetro `irq` se usa en la macro para indexar `irq_table`. La figura 2 ilustra el diseño completo.

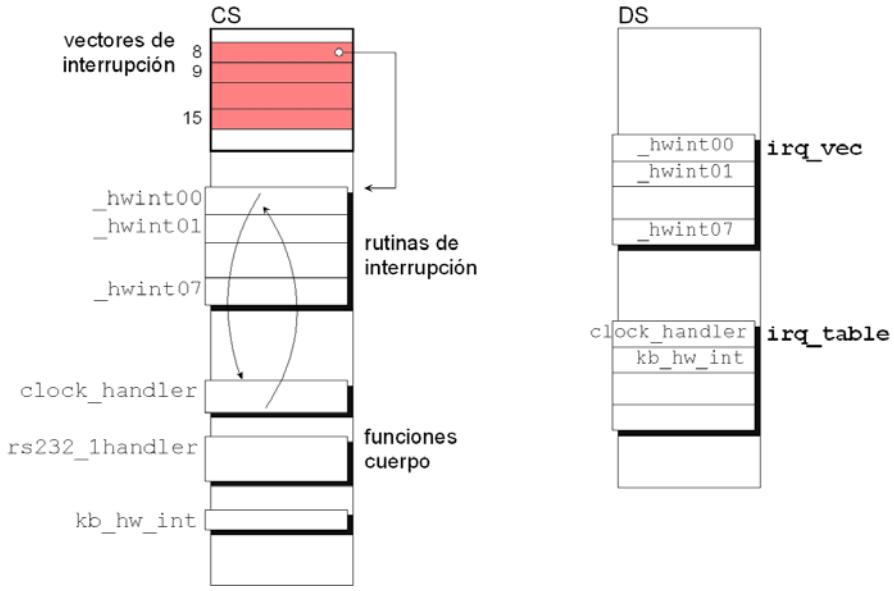

Figura 2. Rutinas de interrupción y funciones cuerpo en MINIX. Sus vectores registro.

Observemos en la figura 2 que la función cuerpo del reloj es `clock_handler`, escrita en */usr/src/kernel/clock.c* y bien conocida ya para nosotros, y que las funciones cuerpo de los puertos RS-232 son `rs232_1handler` y `rs232_2handler` respectivamente, escritas en el fichero */usr/src/kernel/rs232.c*.

2. Retirando a TTY los puertos RS-232 y asignándolos a MOUSE.

La distribución original de Minix considera el RS-232 como un dispositivo más del *driver* del terminal, TTY, como lo es el teclado o la consola de vídeo. Entre otras cosas, eso significa que:

1. Es TTY quien toma la responsabilidad de escribir los **vectores de interrupción** (8 + 3) y (8 + 4) de los puertos RS-232 y de escribir las entradas 3 y 4 del **registro** de funciones cuerpo, la tabla `irq_table` (ver la figura 2). Ambas cosas las lleva a cabo TTY invocando la rutina `put_irq_handler`, invocada por la rutina `rs_init` (en */usr/src/kernel/rs232.c*), invocada a su vez por la rutina de inicialización de TTY, `tty_init`.

2. Es TTY quien se ocupa de invocar el **código de inicialización** de las tarjetas RS-232. Este trabajo también lo realiza `rs_init`.
3. Las funciones cuerpo `rs232_1handler` y `rs232_2handler` envían el mensaje HARD_INT a TTY (En realidad es la tarea del reloj la que indirectamente envía HARD_INT a TTY, pero este es un hecho irrelevante ahora).

En esta práctica nos ocuparemos de modificar `rs232_1handler` para que envíe directamente HARD_INT a MOUSE. De lo que se trata es de retirar a TTY la propiedad de los puertos RS-232 y otorgársela a MOUSE. Para ello:

Modificación 1.
Observemos el código fuente de `rs_init` en el fichero fuente */usr/src/kernel/rs232.c*. Desgraciadamente, está escrita asumiendo que forma parte de TTY. Para empezar tiene un parámetro que es un descriptor de dispositivo TTY. Para eliminar la dependencia de *rs232.c* de TTY, lo más práctico es copiar */usr/src/kernel/rs232.c* en el nuevo fichero */usr/src/kernel/mouse_rs232.c* y modificar este último. Por lo tanto, a partir de ahora el driver del ratón constará de dos ficheros, *mouse.c* y */usr/src/kernel/mouse_rs232.c*. En este último, la nueva `rs_init` se llamará `mouse_rs_init`. La reescribiremos sin parámetro y asumiendo que sólo hay una tarjeta RS-232 en el equipo. Será como sigue:

```
PUBLIC void mouse_rs_init()
{
  register rs232_t *rs;
  port_t            this_8250;
  int               irq;
  long              v;

  line = 0;
  rs   = &rs_lines[0];

  /* Set up input queue. */
  rs->ihead = rs->itail = rs->ibuf;

  /* Precalculate port numbers for speed. Magic numbers in the code (once). */
  this_8250             = addr_8250[0];
  rs->xmit_port         = this_8250 + 0;
  rs->recv_port         = this_8250 + 0;
  rs->div_low_port      = this_8250 + 0;
  rs->div_hi_port       = this_8250 + 1;
  rs->int_enab_port     = this_8250 + 1;
  rs->int_id_port       = this_8250 + 2;
  rs->line_ctl_port     = this_8250 + 3;
  rs->modem_ctl_port    = this_8250 + 4;
  rs->line_status_port  = this_8250 + 5;
  rs->modem_status_port = this_8250 + 6;

  istop(rs);              /* sets modem_ctl_port */
  rs_config(rs);
  out_byte(rs->int_enab_port, 0);

  in_byte(rs->line_status_port);
  in_byte(rs->recv_port);
  rs->ostate = devready(rs) | ORAW | OSWREADY;      /* reads modem_ctl_port */
  rs->ohead = rs->otail = rs->obuf;

  /* Enable interrupts for both interrupt controller and device. */
  irq = RS232_IRQ;
  put_irq_handler(irq, rs232_1handler);
  enable_irq(irq);
  out_byte(rs->int_enab_port, IE_LINE_STATUS_CHANGE | IE_MODEM_STATUS_CHANGE
                           | IE_RECEIVER_READY | IE_TRANSMITTER_READY);

  /* Tell external device we are ready. */
  istart(rs);
}
```

Modificación 2.
Tras modificar `mouse_rs_init`, lo mejor es eliminar de *mouse_rs232.c* todas las funciones que ya no son necesarias para soportar un dispositivo de sólo entrada, a saber, `rs_read`, `rs_write`, `rs_echo`, `rs_ioctl`, `rs_icancel`, `rs_ocancel`, `rs_ostart` y `rs_break`. Tampoco es preciso soportar `rs232_2handler`, ya que sólo hay un ratón en el sistema.

Modificación 3.
En *mouse_rs232.c* vamos a definir el vector `rs_lines` como `static` para no colisionar con el vector del mismo nombre de *rs232.c*.

Modificación 4.
A continuación se muestra el patrón de tarea de E/S Minix.

```
tarea()
{
  mensaje mens;
  int     r, emisor;

  inicializar();
  while(TRUE) {
      receive(ANY, &mens);
      emisor = mens.fuente;
      switch(mens.tipo)
      {
        case READ:     r = do_read();     break;
        case WRITE:    r = do_write();    break;
        case INTERRUP: r = do_interrup(); break;
        case OTRO:     r = do_otro();     break;
        default:       r = ERROR;
      }
      mens.tipo       = TASK_REPLAY;
      mens.REP_STATUS = r;
      send(emisor, &mens);
  }
}
```

Se trata de escribir la función de inicialización de *mouse.c*, `mouse_init`. Esta debe invocar `mouse_rs_init`. En segundo lugar modificaremos *tty.c* para que no inicialice las tarjetas RS_232, una tarea que ya no le compete. Para ello comentaremos la línea de la función `tty_init` que invoca `rs_init`. Finalmente, probaremos que todo va bien. Para ello añadiremos la línea `printf(".");` a la función `rs232_1handler` de *mouse_rs232.c*. La nueva imagen debe mostrar molestos puntitos cuando se mueve el ratón.

3. El buffer de bajo nivel original y el nuevo.

En la práctica previa hemos trabajado el concepto de buffer. En este apartado exploramos el buffer de bajo nivel (BBN) original de los dispositivos RS-232 a fin de sustituirlo por el que ya hemos programado en la práctica previa.

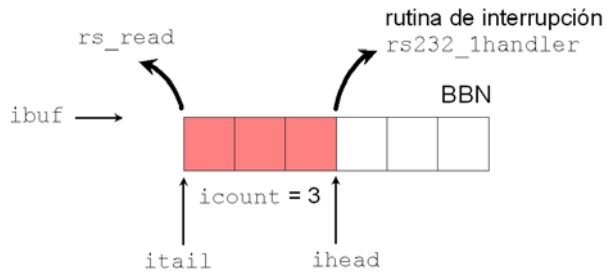

Figura 3. El buffer de bajo nivel original del dispositivo RS_232.

El buffer de bajo nivel original del RS-232 se ilustra en la figura 3. Minix dispone un objeto de tipo `rs232_t` para la gestión de cada uno de los RS-232. El BBN reside en el campo `ibuf` de cada objeto `rs232_t`. Todos los objetos `rs232_t` se reunen en el vector global `rs_lines` según el código que sigue:

```
typedef struct rs232 {
  int    icount;          /* number of bytes in the input buffer */
  char  *ihead;           /* next free spot in input buffer */
  char  *itail;           /* first byte to give to TTY */
  bool_t idevready;       /* nonzero if we are ready to receive (RTS) */
  char   cts;             /* normally 0, but MS_CTS if CLOCAL is set */

  port_t recv_port;
  port_t div_low_port;
  port_t div_hi_port;
  port_t int_enab_port;
  port_t int_id_port;
  port_t line_ctl_port;
  port_t modem_ctl_port;
```

```
  port_t line_status_port;
  port_t modem_status_port;
  unsigned char lstatus;        /* last line status */
  unsigned char pad;            /* ensure alignment for 16-bit ints */
  unsigned framing_errors;      /* error counts (no reporting yet) */
  unsigned overrun_errors;
  unsigned parity_errors;
  unsigned break_interrupts;

  char ibuf[RS_IBUFSIZE]; /* input buffer */
} rs232_t;

static rs232_t rs_lines[NR_RS_LINES];
```

Observemos que el campo `ibuf` es un buffer de caracteres, no de enteros como en la práctica 7. La razón es que el RS-232 proporciona octetos, por supuesto, proporcionados en el registro de datos. El campo `recv_port` es la dirección física del registro de datos de la tarjeta RS-232.

Una vez examinadas las estructuras de datos, vayamos con los métodos. Desgraciadamente, el método *PON* del BBN original se encuentra empotrado en la función `in_int`. La rutina de interrupción del RS-232, `rs232_1handler`, invoca `in_int` para poner en el BBN cada octeto que recoge del registro de datos de la tarjeta vía `in_byte`. Obsérvese que es la rutina de interrupción del reloj la que envía el mensaje HARD_INT a TTY tal y como estudiamos en el tema "El reloj".

```
PRIVATE void in_int(rs)
register rs232_t *rs;           /* line with input interrupt */
{
  int c;

  c = in_byte(rs->recv_port);
  if (rs->icount == buflen(rs->ibuf)) return; /* input buffer full, discard */
  rs->icount++;
  *rs->ihead = c;
  if (++rs->ihead == bufend(rs->ibuf)) rs->ihead = rs->ibuf;
  if (rs->icount == 1) {
       rs->tty->tty_events = 1;
       force_timeout();
  }
}
```

El método QUITA del BBN original se encuentra empotrado en la función `rs_read`, que viene a ser como sigue, un código difícil de comprender y en el que afortunadamente no vamos a entrar:

```
PRIVATE void rs_read()
{
/* Process characters from the circular input buffer. */

  rs232_t *rs = &rs_lines;
  int icount, count, ostate;

  while ((count = rs->icount) > 0) {
       icount = bufend(rs->ibuf) - rs->itail;
       if (count > icount) count = icount;

       /* Perform input processing on (part of) the input buffer. */
       if ((count = in_process(tp, rs->itail, count)) == 0) break;

       lock();                  /* protect interrupt sensitive variables */
       rs->icount -= count;
       if (!rs->idevready && rs->icount < RS_ILOWWATER) istart(rs);
       unlock();
       if ((rs->itail += count) == bufend(rs->ibuf)) rs->itail = rs->ibuf;
  }
}
```

Modificación 5.
Reescribir la función `in_int` con el código PON desarrollado en la práctica anterior y de modo que envíe el mensaje `HARD_INT` a `MOUSE` tras invocar PON. El servicio a HARD_INT debe invocar el método QUITA de BBN para extraer su contenido y mostrarlo en pantalla.