MANEJADOR

RS232

Adai Bujeda Ateca
Guayasén González Santiago

© Universidad de Las Palmas de Gran Canaria

Indice

Introducción ____ *3*

Protocolo RS-232-C ____ *5*

Descripción de la UART ____ *6*

Relación con TTY ____ *11*

Estructura RS-232 ____ *13*

Constantes ____ *15*

Macros ____ *17*

Funciones ____ *18*

rs_write(tp) ____ **18**

rs_echo (tp, c) ____ **26**

rs_ioctl (tp) ____ **28**

rs_config (rs) ____ **21**

in_int(rs) ____ ¡Error! Marcador no definido.

Rs_init (tp) ____ **18**

Rs_icancel (tp) ____ **28**

Rs_ocancel (tp) ____ **28**

Rs_read (tp) ____ **26**

Rs_ostart (rs) ____ **26**

Rs_break (tp) ____ **29**

Rs232_1handler (tp) ____ **29**

Rs232_2handler (tp) ____ **30**

line_int (rs) ____ **34**

modem_int (rs) ____ **34**

out_int (rs) ____ ¡Error! Marcador no definido.

Preguntas ____ *35*

Anexo ____ *36*

Introducción

Este driver es el encargado la transmisión y recepción de datos a través de los puertos serie del ordenador. Gracias a el se soporta la utilización de terminales remotos o la conexión vía módem con otros ordenadores, entre otras cosas. Incluye las funciones de control sobre los dispositivos hardware encargados del manejo de las comunicaciones serie. En el caso de las máquinas tipo IBM PC y compatibles (basadas en la familia de microprocesadores intel), sobre las que se ejecuta el minix, la gestión de los puertos RS-232 se realiza mediante las UARTs (Universal Asynchronous Receiver-Transmitter, receptor-emisor asíncrono universal) 8250 y 16450.

El objetivo último de este driver es proporcionarnos un servicio de comunicación básico mediante una línea RS-232, obviando todo lo referente al hardware específico que estemos utilizando. Cualquier proceso que haga uso de estas funciones (por ejemplo el **tty.c**) puede olvidarse de cuestiones como la programación mediante puertos I/O de la UART, y centrarse en su propio objetivo.

Con el objetivo de facilitar la comprensión del código, se ha incluido una breve descripción de los controladores hardware, donde se detallan principalmente los registros que definen el funcionamiento de los dispositivos.

Un controlador de líneas serie consta, en general, de tres tipos de registros:

- Registros de datos. Generalmente son dos registros, uno de entrada y otro de salida, donde se colocará o de donde se leerá la i que estemos tratando.
- Registros de control. Permiten la programación de la interfase (elección del modo síncrono/asíncrono, velocidad de transmisión Nº de bits de stop, etc...)

- Registros de estado. Este registro contendrá los indicadores de error que se pueden haber producido, tanto en la recepción como en la transmisión de datos.

Protocolo RS-232-C

Es un protocolo diseñado para controlar las comunicaciones serie entre dos terminales.
Las principales líneas que posee son:

- DTR (Data Terminal Ready): indica que el emisor está preparado
- DSR (Data Set Ready): indica que receptor está preparado
- CTS (Clear To Send): Listo para transmitir
- RTS (Request To Send): Petición de envío
- TD (Transmit Data): Por donde se transmiten los datos
- RD (Recieve Data): Por donde se reciven los datos
- CD (Carry Detect): Indica detección de portadora

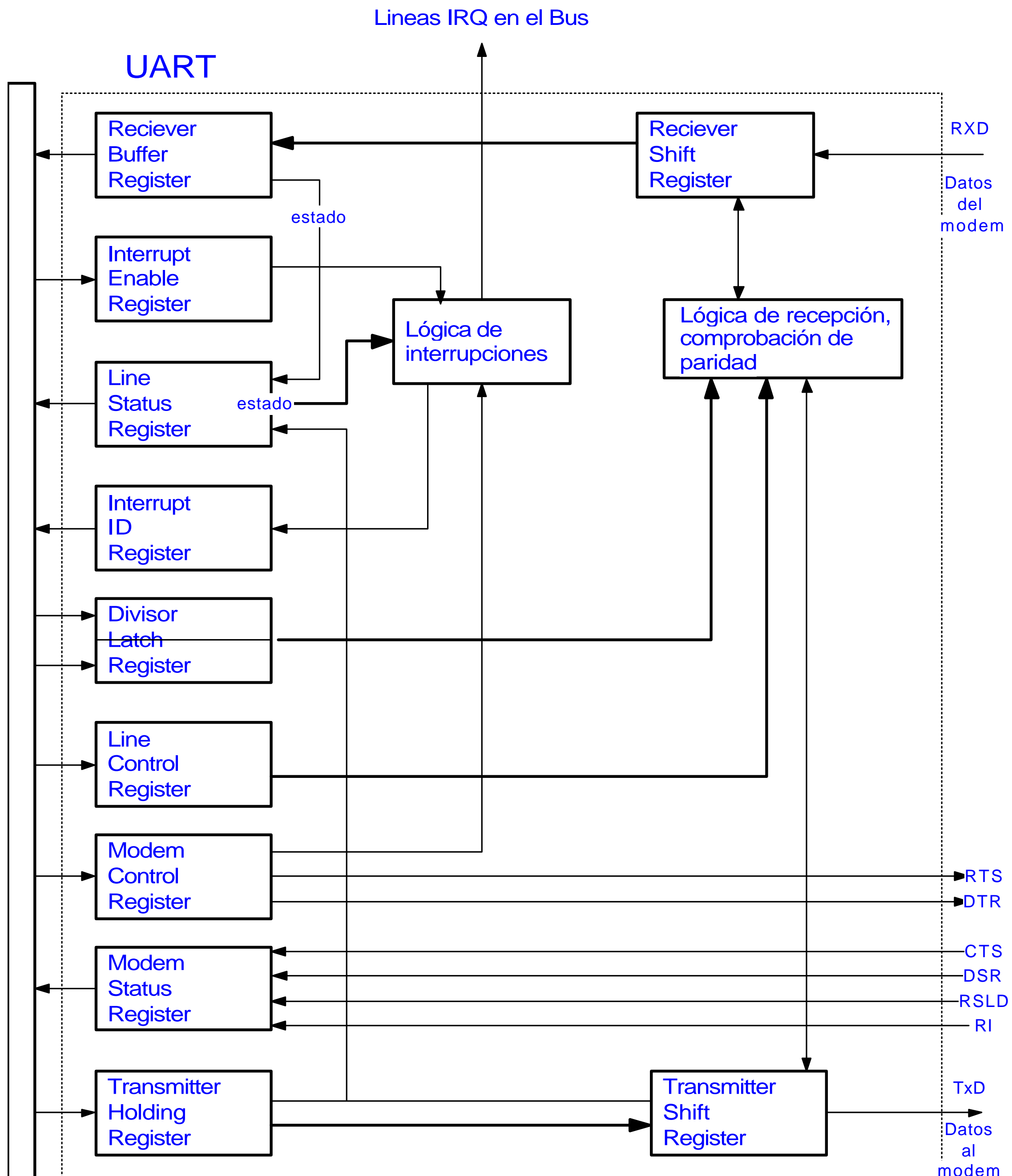
De esta forma:

- DTR-DSR: indica disponibilidad de equipo
- CTS-RTS: indica disponibilidad de canal

DTR
DSR
RTS
CTS
TD

Descripción de la UART

Estudiaremos ahora brevemente el funcionamiento de los registros que especifican el funcionamiento del 8250. El acceso a estos registros en los ordenadores tipo PC y AT es mediante las direcciones de entrada/salida 3F8h y 2F8h (com1 y com2). Cada una de estas direcciones se refiere al registro 0 de la UART que gestiona el puerto serie especificado. El acceso a los demás se realiza sumando el número de registro a esta dirección base. A continuación mostramos un diagrama de la UART



Dirección: 3f8 + 0 h

Regitros buffer de receptor / de transmisor (Reciever Buffer Register, RBR; Transmitter Holding Register, THR). Son dos registros distintos que utilizan la misma dirección física. El chip de la UART distinguirá, mediante el tipo de operación que le indiquemos (leer o escribir al registro), a que registro es el que vamos a acceder.

Regitro Latch divisor de velocidad (Divisor Latch Register, DLR). Lo único que hay que destacar de esta dirección es que cuando el bit 7 del *registro de control de la linea* (DLAB) es TRUE, esta dirección de la UART es el byte menos significativo del registro que indica Latch divisor de velocidad. Esto se verá con más **rs_config**().

Dirección: 3f8 + 1 h

Registro de activación de interrupciones (Interrupt Enable Register, IER). Los bits de este registro se muestran en el siguiente diagrama. Como se observa, este registro permite activar los cuatro tipos de interrupción soportados por la 8250. Cada tipo de interrupción queda activada cuando su bit relativo vale uno, e inhibida si vale cero.

Bit 7 Bit 0

0	0	0	0	MS_CHANGE	LS_CHANGE	TxRDY	RxRDY

MS_CHANGE: Dispara la interrupción en cuanto se modificque el estado del modem en MSR (Modem Status Register).

LS_CHANGE: Dispara la interrupción en cuanto se modifique el estado de la linea en LSR (Line Status Register).

TxRDY: Si está activo, se genera la interrupción tan pronto como la 8250 pueda aceptar otro byte para

RxRDY: Si el bit está a alto, se genera una interrupción siempre que haya un bit disponible para lectura en el registro del buffer del receptor (RBR).

Regitro Latch divisor de velocidad (Divisor Latch Register, DLR). De forma similar a como pasaba en el registro anterior, cuando el bit DLAB del *registro de control de la linea* está a uno, esta dirección pasa a apuntar al byte más significativo del Latch divisor de velocidad.

Dirección: 3f8 + 2 h

Registro de identificación de interrupciones (Interrupt ID Register, IIR). Cuando se detecta una interrupción, se lee este registro para identificar su procedencia exacta. Un 1 en el bit 0 significa que la interrupción ha sido disparada por el puerto serie consultado. Los bits 1 -2 identifican la función según la tabla

siguiente

7					2	1	0	Prioridad	Identificación Interrupción
0	0	0	0	0	x	x	0	Ninguna	Ninguna
0	0	0	0	0	0	0	1	0	Cambio del estado modem
0	0	0	0	0	0	1	1	1	Buffer Transmisión vacío
0	0	0	0	0	1	0	1	2	Dato recibido
0	0	0	0	0	1	1	1	3	Error de serie o Alarma

Nota: 0 tiene la prioridad más alta.

Cambio del estado del modem: se activa cuando alguna de las lineas CTS, DSR, RI o RLSD cambian de estado.

Buffer de transmisión vacío: el registro THR (Transmitter Holding Register) está vacío.

Dato recibido: dato disponible en el RBR (Reciever Buffer Register).

Error de serie o alarma: error durante la transferencia de datos o una alarma.

Dirección: 3f8 + 3 h

Registro de control de la linea (Line Control Register, LCR). En el siguiente diagrama se muestra e l mapa de bits de este registro.

7	6	5 4 3	2	1 0
DLAB	Alarma	Paridad	Stop-Bits	Bits de datos

DLAB (bit de acceso a los registros del Latch divisor), ya ha sido mencionado anteriormente. Este bit no tiene nada que ver con el formato; se trata de un flag que permite extender el número de registros direccionados con las tres líneas de control del chip. Cuando este bit está a uno, las direcciones (base+0) y (base+1) direccionan los registros que contienen el byte menos significativo y el más significativo de los registros de latch divisor.

Alarma: si activo la linea de datos (TxData) se coloca en 0 lógicoindependendientemente del contenido de la s demás líneas (Spacing Condition) para enviarle así al receptor una especie e señal de alarma. La UART vuelve a su modo de funcionamiento normal en cuanto este bit se desactive.

Paridad:

- x x 0 => comprobación y generación de paridad desactivada.
- x x 1 => comprobación y generación de paridad activada.

Y dependiendo de los dos últimos bits:

-0 0 1 => paridad impar.

-0 1 1 => paridad par.

-1 x 1 => paridad constante (0 ó 1) sin relación a los datos.

Si x==1 paridad = 0

Si x==0 paridad = 1

Stop-bits:

0 => 1 stop-bit

1 => 1,5 stop-bit en longitud de datos de 5 y 2 stop-bits en longitud de 6 a 8.

Bits de datos:

00 => 5
01 => 6
10 => 7
11 => 8

Dirección: 3f8 + 4 h
Registro de control del modem (Modem Control Register, MCR). Este registro controla el estado de las dos salidas RS-232 que se incluyen en el chip. Incluye un flag de test local (loopback) y dos salidas de propósito general. Estas funciones escapan al objetivo de este tema, por lo que nos centraremos en los dos primeros bits, que son los ya conocidos RTS y DTR. Hay que hacer notar que estos bits están, en realidad, invertidos, ya que el protocolo RS-232 utiliza lógica negativa. Para nuestros efectos, un uno en cualquiera de estos bits significa "Activo".

7	6	5	4	3	2	1	0
0	0	0	LoopBack	Modo_F		RTS	DTR

DTR: 0 => Linea DTR se desactiva para indicarle al modem que el PC no está listo.
1 => Linea DTR se activa para indicarle al modem que el PC está listo.
RTS: 0 => Linea RTS se desactiva para indicarle al modem que el PC no quiere enviar nada.
1 => Linea RTS se activa para indicarle al modem que el PC quiere enviarle algo.

Modo_F: 0 => la UART no genera interrupciones (modo de funcionamiento haciendo polling)
1 => la UART genera interrupciones en función del IER.
LoopBack: Autotest. Todos los caracteres enviados son recividos por la UART como entrada

Dirección: 3f8 + 5 h
Registro de estado de la linea (Line Status Register, LSR). Este registro informa del estado del proceso de transmisión, incluyendo la detección de una alarma, errores de receptor y actividad de los registros de

Bit 7 Bit 0

0	TXE	TBE	Alarma	Err T	Err P	Err O	RxRDY

Las condiciones de error que informan los bits 1 -4 generan una interrupción cuando está alto el bit del registro de activación de interrupciones.

RxRDY. Este bit está alto cuando ha llegado un byte al registro buffer del receptor (RBR).

Err O (OVERRUN ERROR). Error de sobreescritura en el receptor. Cuando se activa, indica que un byte en el bufer del receptor acaba de ser borrado por la entrada de un nuevo byte.

Err P (PARITY ERROR). Error de paridad. Se activa cuendo el bit de paridad del byte recibido no coincide con la paridad debida según el registro de formato de datos.

Err T (FRAME ERROR). Cuando está alto indica un error de transferencia al recibir el último c aracter

Alarma. Aparece un 1 en este bit cada vez que el receptor detecta que el emisor le envia una alrama.

TBE. Si ==1 indica que el registro buffer del transmisor (THR) está vacío. Se activa para indicar que escribir el siguiente byte (en el THR) que deseemos enviar. (Hay que hacer notar que esto no implica que la transmisión del byte anterior haya sido completada).
Si ==0 indica que el registro buffer del transmisor (THR) contiene un byte que tiene que ser enviado

TXE. Si ==1 indica que el registro shift del transmisor (TSR) está vacío.
Si ==0 indica que el registro shift del transmisor (TSR) contiene un byte que tiene que ser enviado

Dirección: 3f8 + 6 h
Registro de estado del modem (Modem Status Register, MST). Los bits 0-3 de este registro indican si se han producido cambios en el estado de su patilla respectiva RS-232. Un uno en cualquiera de esos bits indica que la entrada ha sido modificada desde la última vez que se leyó.

Los bits 4-7 informan sobre el estado absoluto de sus respectivas entradas RS-232.

Bit 7 Bit 0

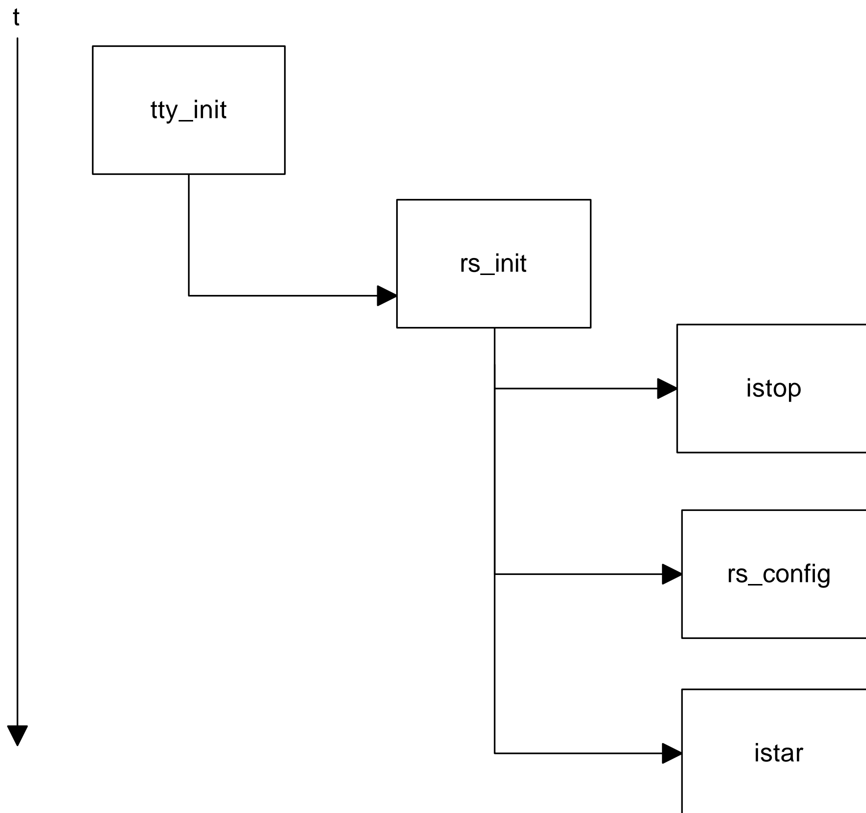
DCD	RI	DSR	CTS	_DCD	_RI	_DSR	_CTS

Relación con TTY

El controlador del terminal se apoya en las funciones que le proporciona el manejador del puerto RS-232 para realizar sus tareas. Para ello, invoca en su inicialización (tty_init) a la función rs_init que se encargará de configurar los parámetros de la comunicación (ver rs_config) y de situar en los campos correspondientes de la estructura TTY los punteros a las funciones de manejo de la línea:

```
tp->tty_devread = rs_read;
tp->tty_devwrite = rs_write;
tp->tty_echo = rs_echo;
tp->tty_icancel = rs_icancel;
tp->tty_ocancel = rs_ocancel;
tp->tty_ioctl = rs_ioctl;
tp->tty_break = rs_break;
```

Seguidamente se expone un diagrama de flujo que refleja el proceso explicado en el parrafo anterior:



Código de la función TTY_init:

```
PRIVATE void tty_init(tp)
tty_t *tp;
{
  tp->tty_intail = tp->tty_inhead = tp->tty_inbuf;
  tp->tty_min = 1;
  tp->tty_termios = termios_defaults;
  tp->tty_icancel = tp->tty_ocancel = tp->tty_ioctl = tp->tty_close =
tty_devnop;
  if (tp < tty_addr(NR_CONS)) {
 scr_init(tp);
  } else
  if (tp < tty_addr(NR_CONS+NR_RS_LINES)) {
 rs_init(tp);
 } else {
 pty_init(tp);
  }
}
```

Estructura RS-232

A continuación, estudiaremos el fragmento de código donde están declaradas la mayoría de las variables y estructuras con las que vamos a trabajar. La estructura de datos más utilizada es la **struct rs232_s**, que incluye todas las variables y parámetros necesarios para el control del puerto RS-232. Habrá una estructura rs232_s para cada puerto serie (2 por defecto).

```
/* Types. */
typedef unsigned char bool_t; /* boolean */

/* RS232 device structure, one per device. */
typedef struct rs232 {
  tty_t *tty;                /* associated TTY structure */

  int icount;
   número de BYTES en el bufer de entrada
  char *ihead;
   apuntador al siguiente hueco libre en el buffer de entrada
  char *itail;
   apuntador al primer byte a dar a la tty en el buffer de entrada
  bool_t idevready;
   si es == 0 indica que estamos preparados para recibir RTS
  char cts;
   normalmente == 0, pero == MS_CTS si CLOCAL está activo
  unsigned char ostate;
   combinación de los siguientes flags
#define ODONE          1
salida completada
#define ORAW           2
RAW mode para xoff deshabilitado
#define OWAKEUP        4
pendiente de tty_wakeup()
#define ODEVREADY MS_CTS
hardware del dispositivo externo (modem) preparado (CTS)
#define OQUEUED     0x20
buffer de salida no está vacio
#define OSWREADY    0x40
software del dispositivo externo preparado (no xoff)
#define ODEVHUP  MS_RLSD
dispositivo externo ha parado la portadora
#define OSOFTBITS  (ODONE | ORAW | OWAKEUP | OQUEUED | OSWREADY)
         /* user-defined bits */
#if (OSOFTBITS | ODEVREADY | ODEVHUP) == OSOFTBITS
         /* a weak sanity check */
#error           /* bits are not unique */
#endif
  unsigned char oxoff;
   caracter para xoff (control de flujo software)
  bool_t inhibited;          /* output inhibited? (follows
tty_inhibited) */
  bool_t drain;
   indica que hay que vaciar el buffer de salida => para reconfigurar
el puerto


 int ocount;
 número de BYTES en el bufer de salida
 char *ohead;
```

```
 apuntador al siguiente hueco libre en el buffer de salida
 char *otail;
 apuntador al primer byte a dar a la tty en el buffer de salida
#if (MACHINE == IBM_PC)
se definen los puertos de la UART
  port_t xmit_port;           transmisión
  port_t recv_port;           recepción
  port_t div_low_port;        latch divisor
  port_t div_hi_port;         latch divisor
  port_t int_enab_port;       habilitación de interrupciones
  port_t int_id_port;         identificación de interrupciones
  port_t line_ctl_port;       control de la linea
  port_t modem_ctl_port;      control del modem
  port_t line_status_port;    estado de la linea
  port_t modem_status_port;   estado del modem
#endif

  unsigned char lstatus;
  último lines status
  unsigned char pad;          /* ensure alignment for 16-bit ints */

  contadores de los errores (no se informa todavia)
  unsigned framing_errors;
  unsigned overrun_errors;
  unsigned parity_errors;
  unsigned break_interrupts;

  char ibuf[RS_IBUFSIZE];     /* input buffer */
  char obuf[RS_OBUFSIZE];     /* output buffer */
} rs232_t;
```

La inicialización del espacio en memoria para el array de estructuras que corresponden a cada puerto RS-232 se realiza en la siguiente declaración.

PUBLIC rs232_t rs_lines[NR_RS_LINES];

Constantes

Las constantes que utilizaremos como parámetros por defecto son **DEF_BAUD**, que nos indicará la velocidad en baudios inicial, 1200 baudios. Para transformar la velocidad en baudios a los bytes del latch divisor de velocidad (Registros 0 y 1 con DLAB activo) se necesita la formula

$$Divisor = \frac{Freq.referencia}{16xVelocidad(baudios)}$$

La frecuencia de referencia de la anterior fórmula está definida en la constante **UART_FREQ**.

La siguiente tabla almacena las direcciones de comienzo de los registros 0 de cada puerto serie. Para acceder a los restantes se añade a esta dirección el valor correspondiente.

```
PRIVATE port_t addr_8250[] = {
  0x3F8,           /* COM1: (línea 0);COM3 debe estar en 0x3E8 */
  0x2F8,           /* COM2: (línea 1);COM4 debe estar en 0x2E8 */
};
```

Las siguientes definiciones especifican la tabla y la macro necesarios para traducir el número del puerto serial a un puntero a su correspondiente estructura rs232_s.

```
struct rs232_t *p_rs_addr[NR_RS_LINES];
#define rs_addr(line) (p_rs_addr[line])
```

NR_RS_LINES es una variable definida en tty.h, que indica el número de terminales por defecto que tiene el sistema. Por defecto vale 2.

Las siguientes declaraciones de constantes van a ser usadas como máscaras para los diferentes registros de la 8250.

```
/* 8250 constants. */
#define UART_FREQ         115200L   /* timer frequency */

/* Interrupt enable bits. */
#define IE_RECEIVER_READY       1
/* habilita la interrupcion receptor listo */
#define IE_TRANSMITTER_READY    2
/* habilita la interrupcion de que se puede enviar otro carácter */
#define IE_LINE_STATUS_CHANGE   4
/* habilita interrupcion en cuanto se detecte cambio en LSR */
#define IE_MODEM_STATUS_CHANGE  8
/* habilita interrupcion en cuanto se detecte cambio en MSR */


/* Interrupt status bits. */
#define IS_MODEM_STATUS_CHANGE  0 /* modem ha cambiado */
#define IS_TRANSMITTER_READY    2 /* Transmisor listo */
#define IS_RECEIVER_READY       4 /* receptor listo */
```

```
#define IS_LINE_STATUS_CHANGE   6 /* cambio en el LSR */

/* Line control bits. */
#define LC_2STOP_BITS        0x04   /* 2 stops bits */
#define LC_PARITY            0x08  /*paridad activa */
#define LC_PAREVEN           0x10  /* paridad par  */
#define LC_BREAK             0x40  /* señal de alarma  */
#define LC_ADDRESS_DIVISOR   0x80  /* indica si está activo, que RBR y
IER contienen el Latch divisor de velocidad */

/* Line status bits. */
#define LS_OVERRUN_ERR          2 /* error de overrun */
#define LS_PARITY_ERR           4 /* error de paridad */
#define LS_FRAMING_ERR          8 /* error en la trama */
#define LS_BREAK_INTERRUPT   0x10
#define LS_TRANSMITTER_READY 0x20

/* Modem control bits. */
#define MC_DTR                  1
#define MC_RTS                  2
#define MC_OUT2                 8
/* indica que el control de la UART es por interrupciones no por
polling*/

/* Modem status bits. */
#define MS_CTS               0x10
#define MS_RLSD              0x80       /* Received Line Signal Detect */
#define MS_DRLSD             0x08       /* RLSD Delta */


#define DATA_BITS_SHIFT         8   /* amount data bits shifted in mode
*/
#define DEF_BAUD             1200   /* default baud rate */

#define RS_IBUFSIZE          1024   /* RS232 input buffer size */
#define RS_OBUFSIZE          1024   /* RS232 output buffer size */

/* Input buffer watermarks.
 * The external device is asked to stop sending when the buffer
 * exactly reaches high water, or when TTY requests it.  Sending restarts
 * when the input buffer empties below the low watermark.
 */
#define RS_ILOWWATER    (1 * RS_IBUFSIZE / 4)
#define RS_IHIGHWATER   (3 * RS_IBUFSIZE / 4)

/* Output buffer low watermark.
 * TTY is notified when the output buffer empties below the low
watermark, so
 * it may continue filling the buffer if doing a large write.
 */
#define RS_OLOWWATER    (1 * RS_OBUFSIZE / 4)
```

Macros

A continuación estudiaremos las funciones de bajo nivel que serán utilizadas por todos los niveles superiores. Las dos primeras que estudiaremos, **istart()** y **istop()** gestionan el control del flujo de datos, mediante el protocolo RTS-CTS. Las funciones **out_byte()** e **in_byte()**, que se utilizarán a menudo en las restantes definiciones, se encuentran en el fichero klib88.x

```
#define istart(rs) \
  (out_byte( (rs)->modem_ctl_port, MC_OUT2 | MC_RTS | MC_DTR),
(rs)->idevready = TRUE)
#define istop(rs) \
  (out_byte( (rs)->modem_ctl_port, MC_OUT2 | MC_DTR), (rs)->idevready =
FALSE)
```

Devready() nos informa si el dispositivo esta preparado:

```
#define devready(rs) ((in_byte(rs->modem_status_port) | rs->cts) &
MS_CTS)
/* devuelve el estado del registro del modem poniendo activo el bit cts
*/
```

La siguiente macro informa si estamos preparados para enviar. Si no utilizáramos **txready()**, podríamos empezar a transmitir un byte sin que hubiera terminado la transmisión del anterior (ver el flag

```
#define txready(rs) (in_byte(rs->line_status_port) &
LS_TRANSMITTER_READY)
```

Devhup() nos dice si se ha perdido la portadora:

```
#define devhup(rs)      \
  (in_byte(rs->modem_status_port) & (MS_RLSD|MS_DRLSD) == MS_DRLSD)
```

Funciones

Rs_init (tp)

Inicializa una línea rs_232.
Realiza la asociación entre las estructuras rs-232 y TTY.
Inicializa el buffer de entrada.
Asigna los puertos de la UART a variables.
Baja la RTS (istop) y llama a *rs_config* para configurar el rs-232.
Inhibe las interrupciones.
Inicializa el buffer de salida.
Comprueba que el driver de red no utilice la misma IRQ que la rs-232.
Asignamos las funciones propias de la rs-232 a la estructura TTY para que puedan ser llamadas desde tty.c .
Subimos la RTS (istart) para indicar que estamos preparados.

```
PUBLIC void rs_init(tp)
tty_t *tp;              /* que TTY */
{
/* Inicializa la RS232 para un puerto serie. */

  register rs232_t *rs;
  int line;
#if (MACHINE == IBM_PC)
  port_t this_8250;
  int irq;
  long v;
#endif

  /* Asocia las estructuras rs-232 y la tty */
  line = tp - &tty_table[NR_CONS];
  rs = tp->tty_priv = &rs_lines[line];
  rs->tty = tp;

  /* Inicializa el buffer de entrada. (cola circular) */
  rs->ihead = rs->itail = rs->ibuf;

#if (MACHINE == IBM_PC)
/* Se calculan las direcciones de los registros de la UART*/
  this_8250 = addr_8250[line];
  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;
#endif

/* Set up the hardware to a base state, in particular
   * o turn off DTR (MC_DTR) to try to stop the external device.
```

```
   * o be careful about the divisor latch.  Some BIOS's leave it
enabled
   *   here and that caused trouble (no interrupts) in version 1.5 by
   *   hiding the interrupt enable port in the next step, and worse
trouble
   *   (continual interrupts) in an old version by hiding the receiver
   *   port in the first interrupt.  Call rs_ioctl() early to avoid
this.
   * o disable interrupts at the chip level, to force an edge
transition
   *   on the 8259 line when interrupts are next enabled and active.
   *   RS232 interrupts are guaranteed to be disabled now by the 8259
   *   mask, but there used to be trouble if the mask was set without
   *   handling a previous interrupt.
   */
  istop(rs);      /* desactivamos el RTS (PC no quiere enviar nada)*/
/* configuramos el puerto serie */
  rs_config(rs);
#if (MACHINE == IBM_PC)
  out_byte(rs->int_enab_port, 0); /* deshabilita las interrupciones */
#endif
  /* Clear any harmful leftover interrupts.  An output interrupt is
harmless
   * and will occur when interrupts are enabled anyway.  Set up the
output
   * queue using the status from clearing the modem status interrupt.
   */
#if (MACHINE == IBM_PC)
  /* Inicializa registros de la UART */
  in_byte(rs->line_status_port);
  in_byte(rs->recv_port);
#endif
  rs->ostate = devready(rs) | ORAW | OSWREADY; /* reads modem_ctl_port
*/

/* Inicializa el buffer de salida. (cola circular) */
  rs->ohead = rs->otail = rs->obuf;

#if (MACHINE == IBM_PC)
/* Alamcena en irq el nº de interrupción del puerto que estemos
   tratando. */
  irq = (line & 1) ? SECONDARY_IRQ : RS232_IRQ;

#if ENABLE_NETWORKING
   /* La interfaz de red puede tener la irq de la rs232 */
  v = ETHER_IRQ;
  switch (env_parse("DPETH0", "x:d:x", 1, &v, 0L, (long)
NR_IRQ_VECTORS-1)) {
  case EP_ON:
  case EP_SET:
   if (v == irq) return;      /* IRQ en uso, no configurar la linea */
  }
#endif
  /* ponemos en la tabla de interrupciones la rutina de manejo*/
  put_irq_handler(irq, (line & 1) ? rs232_2handler : rs232_1handler);
  /* Habilitamos las interrupciones */
  enable_irq(irq);
  out_byte(rs->int_enab_port, IE_LINE_STATUS_CHANGE |
IE_MODEM_STATUS_CHANGE
          | IE_RECEIVER_READY | IE_TRANSMITTER_READY);
#else /* MACHINE == ATARI */
```

```
  /* Initialize the 68901 chip, then enable interrupts. */
  MFP->mf_scr = 0x00;
  MFP->mf_tcdcr |= T_Q004;
  MFP->mf_rsr = R_ENA;
  MFP->mf_tsr = T_ENA;
  MFP->mf_aer = (MFP->mf_aer | (IO_SCTS|IO_SDCD)) ^
      (MFP->mf_gpip & (IO_SCTS|IO_SDCD));
  MFP->mf_ddr = (MFP->mf_ddr & ~ (IO_SCTS|IO_SDCD));
  MFP->mf_iera |= (IA_RRDY|IA_RERR|IA_TRDY|IA_TERR);
  MFP->mf_imra |= (IA_RRDY|IA_RERR|IA_TRDY|IA_TERR);
  MFP->mf_ierb |= (IB_SCTS|IB_SDCD);
  MFP->mf_imrb |= (IB_SCTS|IB_SDCD);
#endif /* MACHINE == ATARI */

/* Rellenamos en la estructura TTY las funciones de bajo nivel */
  tp->tty_devread = rs_read;
  tp->tty_devwrite = rs_write;
  tp->tty_echo = rs_echo;
  tp->tty_icancel = rs_icancel;
  tp->tty_ocancel = rs_ocancel;
  tp->tty_ioctl = rs_ioctl;
  tp->tty_break = rs_break;

  /* Activa el CTS (preparados para transmitir/recibir)*/
  istart(rs);
}
```

rs_config (rs)

Configura los parámetros de control de la línea rs-232

- obtener los valores a situar en los registros 0 y 1 (LDR) para configurar la velocidad.
- configurar el caracter “xoff”.
- comprobación del correcto funcionamiento de la CTS.
- Ajustar la velocidad con los valores obtenidos en el paso 1.
- Ajustar la paridad y número de stop bits.
- Escritura de todos los valores en los registos de la UART.

```
PRIVATE void rs_config(rs)
rs232_t *rs;                            /* which line */
{
/* Set various line control parameters for RS232 I/O.
 * If DataBits == 5 and StopBits == 2, 8250 will generate 1.5 stop
bits.
 * The 8250 can't handle split speed, so we use the input speed.
 */

  tty_t *tp = rs->tty;
  int divisor;
  int line_controls;
  static struct speed2divisor {
   speed_tspeed;
   int divisor;
  } s2d[] = {
#if (MACHINE == IBM_PC)
   { B50, UART_FREQ / 50                },
#endif
   { B75, UART_FREQ / 75                },
   { B110,    UART_FREQ / 110           },
   { B134,    UART_FREQ * 10 / 1345   },
   { B150,    UART_FREQ / 150           },
   { B200,    UART_FREQ / 200           },
   { B300,    UART_FREQ / 300           },
   { B600,    UART_FREQ / 600           },
   { B1200,   UART_FREQ / 1200        },
#if (MACHINE == IBM_PC)
   { B1800,   UART_FREQ / 1800        },
#endif
   { B2400,   UART_FREQ / 2400        },
   { B4800,   UART_FREQ / 4800        },
   { B9600,   UART_FREQ / 9600        },
   { B19200,  UART_FREQ / 19200       },
```

```
#if (MACHINE == IBM_PC)
   { B38400,  UART_FREQ / 38400       },
   { B57600,  UART_FREQ / 57600       },
   { B115200, UART_FREQ / 115200L     },
#endif
  };
  struct speed2divisor *s2dp;

/* rs232 necesita saber cual es el carácter xoff y si el CTS
     funciona (se usa)*/
  rs->oxoff = tp->tty_termios.c_cc[VSTOP];
  rs->cts = (tp->tty_termios.c_cflag & CLOCAL) ? MS_CTS : 0;

/* Configuramos la velocidad del puerto */
  divisor = 0;
  for (s2dp = s2d; s2dp < s2d + sizeof(s2d)/sizeof(s2d[0]); s2dp++) {
   if (s2dp->speed == tp->tty_termios.c_ospeed) divisor = s2dp-
>divisor;
  }
  if (divisor == 0) return;          /* B0? */

#if (MACHINE == IBM_PC)
  /* Compute line control flag bits. */
  line_controls = 0;
  /* configuramos la paridad */
  if (tp->tty_termios.c_cflag & PARENB) {
   line_controls |= LC_PARITY;  /* paridad impar */
   if (!(tp->tty_termios.c_cflag & PARODD)) line_controls |=
LC_PAREVEN; /* paridad par */
  }
   /* define los stop-bits */
  if (divisor >= (UART_FREQ / 110)) line_controls |= LC_2STOP_BITS;
  /* define los bits de datos */
  line_controls |= (tp->tty_termios.c_cflag & CSIZE) >> 2;

  /* Lock out interrupts while setting the speed. The receiver
register is
   * going to be hidden by the div_low register, but the input
interrupt
   * handler relies on reading it to clear the interrupt and avoid
looping
   * forever.
   */
  lock();
```

```
/* Selecionamos el LDR y configuramos la velocidad */
  out_byte(rs->line_ctl_port, LC_ADDRESS_DIVISOR);
  out_byte(rs->div_low_port, divisor);
  out_byte(rs->div_hi_port, divisor >> 8);

/* Configurar la linea de datos */
  out_byte(rs->line_ctl_port, line_controls);

  rs->ostate |= ORAW;
  if ((tp->tty_termios.c_lflag & IXON) && rs->oxoff !=
_POSIX_VDISABLE)
   rs->ostate &= ~ORAW;

  unlock();

#else /* MACHINE == ATARI */

  line_controls = U_Q16;
  if (tp->tty_termios.c_cflag & PARENB) {
   line_controls |= U_PAR;
   if (!(tp->tty_termios.c_cflag & PARODD)) line_controls |= U_EVEN;
  }
  line_controls |= (divisor >= (UART_FREQ / 110)) ? U_ST2 : U_ST1;

  switch (tp->tty_termios.c_cflag & CSIZE) {    /* XXX - are U_Dn like
CSn? */
   case CS5:  line_controls |= U_D5; break;
   case CS5:  line_controls |= U_D6; break;
   case CS5:  line_controls |= U_D7; break;
   case CS5:  line_controls |= U_D8; break;
  }
  lock();
  MFP->mf_ucr = line_controls;
  MFP->mf_tddr = divisor;
  unlock();
#endif /* MACHINE == ATARI */
}
```

rs_write(tp)

Si la “inhibición” ha cambiado entonces
    Se actualiza la variable de estado (ostate), reflejando el actual estado de OSWREADY o no.
    Actualiza la variable interna de “inhibido”.
Finsi
Si se quiere reconfigurar la línea entonces
    Se debe esperar a que se vacíe la salida.
Finsi
Mientras haya algo que hacer (bucle “infinito”)
    Si no queda nada por enviar o el tty está inhibido entonces
        Retornar.
    Finsi
    Copia los datos al buffer de salida.
    Realiza el proceso de salida en el buffer de salida.
    Si no quedan datos entonces
        Pasa a otra iteración del bucle para comprobar si han llegado mas datos.
    Finsi
    Actualiza los contadores y los punteros al buffer de salida.
Finmientras

```
PRIVATE void rs_write(tp)
register tty_t *tp;
{
/* (*devwrite)()0 routine for RS232. */

  rs232_t *rs = tp->tty_priv;
  int count, ocount;
  phys_bytes user_phys;

  if (rs->inhibited != tp->tty_inhibited) {
   /* El estado de inhibición ha cambiado. */
   lock();
   rs->ostate |= OSWREADY;
   if (tp->tty_inhibited) rs->ostate &= ~OSWREADY;
   unlock();
   rs->inhibited = tp->tty_inhibited;
   }

  if (rs->drain) {
/* Esperar para vaciar el buffer de salida para reconfigurar la linea*/
   if (rs->ocount > 0) return;
   rs->drain = FALSE;
   rs_config(rs);
   }
```

```
  /* Mientras haya algo que hacer */
  for (;;) {
   /* ocount = espacio libre del buffer (mirando indices)*/
   /* count = espacio libre del buffer (mirando direcciones)*/
   ocount = buflen(rs->obuf) - rs->ocount;
   count = bufend(rs->obuf) - rs->ohead;
   if (count > ocount) count = ocount;
   if (count > tp->tty_outleft) count = tp->tty_outleft;
   if (count == 0 || tp->tty_inhibited) return;

   /* Copia de la zona de usuario al buffer de salida de la rs232*/
   user_phys = proc_vir2phys(proc_addr(tp->tty_outproc), tp-
>tty_out_vir);
   phys_copy(user_phys, vir2phys(rs->ohead), (phys_bytes) count);

   /* Perform output processing on the output buffer. */
   out_process(tp, rs->obuf, rs->ohead, bufend(rs->obuf), &count,
&ocount);
   if (count == 0) break;

   /* Hay que repintar la tty */
   tp->tty_reprint = TRUE;

   /* Bookkeeping. */
   lock();        /* protect interrupt sensitive rs->ocount */
   rs->ocount += ocount;
   /* iniciamos la transferencia física */
   rs_ostart(rs);
   unlock();
   /* actualiza los punteros del buffer */
   if ((rs->ohead += ocount) >= bufend(rs->obuf))
     rs->ohead -= buflen(rs->obuf);
  /* actualiza los punteros de la tty */
   tp->tty_out_vir += count;
   tp->tty_outcum += count;
   if ((tp->tty_outleft -= count) == 0) {
     /* si se acabo transmitir avisar al usuario*/
     tty_reply(tp->tty_outrepcode, tp->tty_outcaller,
            tp->tty_outproc, tp->tty_outcum);
     tp->tty_outcum = 0;
   }
  }
}
```

Rs_ostart (rs)

Le dice al rs232 que hay algo esperando en el buffer de salida.

```
PRIVATE void rs_ostart(rs)
rs232_t *rs;                  /* which rs line */
{
/* Tell RS232 there is something waiting in the output buffer. */

  rs->ostate |= OQUEUED;
  if (txready(rs)) out_int(rs);
}
```

Rs_read (tp)

Si se detecta que no hay conexión (se perdió la portadora)
  Actualizar el estado (ostate).
  Notificar el cambio mediante una señal.
Finsi
Mientras hayan datos en el buffer
    Leer del buffer.
    Actualizar contadores y punteros.
    Si la (RTS ha sido bajada y el buffer está bajo mínimos)
        Subir la RTS para seguir recibiendo datos.
    Finsi
Finmientras

```
PRIVATE void rs_read(tp)
tty_t *tp;              /* which tty */
{
/* Process characters from the circular input buffer. */

  rs232_t *rs = tp->tty_priv;
  int icount, count, ostate;

  if (!(tp->tty_termios.c_cflag & CLOCAL)) {
   /* Enviar la señal de SIGHUP si se perdio la conexión*/
   lock();
   ostate = rs->ostate;
   rs->ostate &= ~ODEVHUP;           /* save ostate, clear DEVHUP */
   unlock();
   if (ostate & ODEVHUP) { sigchar(tp, SIGHUP); return; }
  }

  while ((count = rs->icount) > 0) {
   /* icount = bytes que hay en la cola (mirando direcciones) */
   /* count = bytes que hay en la cola  (mirando indices)*/
   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 */
  /* actualizamos los punteros */
  rs->icount -= count;
/* si el dispositivo (nosotros) está listo y el nº de bytes en el
buffer < RS_ILOWWATER => activar CTS */
  if (!rs->idevready && rs->icount < RS_ILOWWATER) istart(rs);
  unlock();
  if ((rs->itail += count) == bufend(rs->ibuf)) rs->itail = rs->ibuf;
  }
}
```

rs_echo (tp, c)

Como rs_write, pero caracter a caracter.

```
PRIVATE void rs_echo(tp, c)
tty_t *tp;                              /* which TTY */
int c;       /* character to echo */
{
/* Echo one character.  (Like rs_write, but only one character,
optionally.) */

  rs232_t *rs = tp->tty_priv;
  int count, ocount;
   /* ocount = espacio libre en el buffer */
  ocount = buflen(rs->obuf) - rs->ocount;
  if (ocount == 0) return;              /* output buffer full */
  /* Solo un carácter */
  count = 1;
  *rs->ohead = c;                               /* add one character */

  out_process(tp, rs->obuf, rs->ohead, bufend(rs->obuf), &count,
&ocount);
  if (count == 0) return;

  lock();
  rs->ocount += ocount;
  rs_ostart(rs);
  unlock();
/* actualizamos los punteros */
  if ((rs->ohead += ocount) >= bufend(rs->obuf)) rs->ohead -=
buflen(rs->obuf);
}
```

Rs_icancel (tp)

Cancela la entrada pendiente vaciando el buffer.

```
PRIVATE void rs_icancel(tp)
tty_t *tp;              /* which TTY */
{
/* Cancel waiting input. */
  rs232_t *rs = tp->tty_priv;

  lock();
  rs->icount = 0;
  rs->itail = rs->ihead;
  istart(rs);
  unlock();
}
```

Rs_ocancel (tp)

Cancela la salida pendiente vaciando el buffer.

```
PRIVATE void rs_ocancel(tp)
tty_t *tp;              /* which TTY */
{
/* Cancel pending output. */
  rs232_t *rs = tp->tty_priv;

  lock();
  rs->ostate &= ~(ODONE | OQUEUED);
  rs->ocount = 0;
  rs->otail = rs->ohead;
  unlock();
}
```

rs_ioctl (tp)

Actualiza un flag para indicar que se quiere reconfigurar la línea

```
PRIVATE void rs_ioctl(tp)
tty_t *tp;                              /* which TTY */
{
/* Reconfigure the line as soon as the output has drained. */
  rs232_t *rs = tp->tty_priv;

  rs->drain = TRUE;
}
```

Rs_break (tp)

Genera una alarma durante 0'4 segundos usando el bit Break.

```
PRIVATE void rs_break(tp)
tty_t *tp;             /* which tty */
{
/* Genera una condición alarma  durante 0.4 segundos */
  rs232_t *rs = tp->tty_priv;
  int line_controls;

  line_controls = in_byte(rs->line_ctl_port);
  out_byte(rs->line_ctl_port, line_controls | LC_BREAK);
  milli_delay(400);                                 /* ouch */
  out_byte(rs->line_ctl_port, line_controls);
}
```

Rs232_1handler (tp)

Manejador de interrupciones del COM1.

```
PRIVATE int rs232_1handler(irq)
int irq;
{
/* Manejador de interrupciones para IRQ4.
 * Solo unal inea (normalmente COM1) deberia de usarlo */

  register rs232_t *rs = &rs_lines[0];

  while (TRUE) {
   /* Loop to pick up ALL pending interrupts for device.
    * This usually just wastes time unless the hardware has a buffer
    * (and then we have to worry about being stuck in the loop too
long).
    * Unfortunately, some serial cards lock up without this.
    */
   switch (in_byte(rs->int_id_port)) {
   case IS_RECEIVER_READY:
/* llego un nuevo byte a la UART (registro RBR)
     in_int(rs);
     continue;
   case IS_TRANSMITTER_READY:
  /* El THR se ha vaciado. (registro del transmisor) */
     out_int(rs);
     continue;
   case IS_MODEM_STATUS_CHANGE:
  /* Cambio del estado de las líneas del módem (CTS, DSR, RI y RLSD)*/
     modem_int(rs);
     continue;
   case IS_LINE_STATUS_CHANGE:
  /* Error durante la transferencia de datos o alarma */
     line_int(rs);
     continue;
   }
   return(1);      /* reenable serial interrupt */
  }
}
```

Rs232_2handler (tp)

Identifica el tipo de interrupción en COM2.

```
PRIVATE int rs232_1handler(irq)
int irq;
{
/* Manejador de interrupciones para IRQ4.
 * Solo unal inea (normalmente COM1) deberia de usarlo */

register rs232_t *rs = &rs_lines[1];

  while (TRUE) {
   /* Loop to pick up ALL pending interrupts for device.
    * This usually just wastes time unless the hardware has a buffer
    * (and then we have to worry about being stuck in the loop too
long).
    * Unfortunately, some serial cards lock up without this.
    */
   switch (in_byte(rs->int_id_port)) {
   case IS_RECEIVER_READY:
/* llego un nuevo byte a la UART (registro RBR)
     in_int(rs);
     continue;
   case IS_TRANSMITTER_READY:
  /* El THR se ha vaciado. (registro del transmisor) */
     out_int(rs);
     continue;
   case IS_MODEM_STATUS_CHANGE:
  /* Cambio del estado de las líneas del módem (CTS, DSR, RI y RLSD)*/
     modem_int(rs);
     continue;
   case IS_LINE_STATUS_CHANGE:
  /* Error durante la transferencia de datos o alarma */
     line_int(rs);
     continue;
   }
   return(1);      /* reenable serial interrupt */
   }
}
```

```
PRIVATE void in_int(rs)
register rs232_t *rs;         /* line with input interrupt */
{
/* Read the data which just arrived.
 * If it is the oxoff char, clear OSWREADY, else if OSWREADY was
clear, set
 * it and restart output (any char does this, not just xon).
 * Put data in the buffer if room, otherwise discard it.
 * Set a flag for the clock interrupt handler to eventually notify
TTY.
 */

  int c;

#if (MACHINE == IBM_PC)
/* Lee el carácter del registro RBR de la Uart */
  c = in_byte(rs->recv_port);
#else /* MACHINE == ATARI */
  c = MFP->mf_udr;
#endif

  if (!(rs->ostate & ORAW)) { /* si control de flujo software */
   if (c == rs->oxoff) { /*si recibimos el carácter XOFF */
     rs->ostate &= ~OSWREADY;   /* ponemos el estado a not OSWREADY */
   } else
   if (!(rs->ostate & OSWREADY)) { /* si el estado es not OSWREADY */
     rs->ostate |= OSWREADY;    /* lo ponemos como OSWREADY */
     if (txready(rs)) out_int(rs);
/* si estamos preparados para transmitir, lo intentamos */
   }
 }
```

```
/* Si el buffer de entrada está lleno retornamos */
  if (rs->icount == buflen(rs->ibuf)) return;

if (++rs->icount == RS_IHIGHWATER && rs->idevready) istop(rs);
/* si rebasamos el umbral de seguridad para el buffer de entrada, bajamos la señal RTS*/

   rs->ihead = c; /* guardamos el carácter */
/*actualizamos los punteros del buffer */
  if (++rs->ihead == bufend(rs->ibuf)) rs->ihead = rs->ibuf;
 if (rs->icount == 1) {
/* si sólo hay un carácter en el buffer forzamos el timeout del reloj */
     rs->tty->tty_events = 1;
     force_timeout();
  }
}
```

out_int (rs)

Si el dispositivo (modem) esta preparado, existen datos y el software externo está preparado entonces

> Envía un byte a la UART.
>
> Actualiza los punteros y los contadores del buffer de salida.
>
> Si el buffer está "suficientemente" vacio, se le notifica a la TTY para que lo siga llenando.

Finsi

```
PRIVATE void out_int(rs)
register rs232_t *rs;                    /* line with output interrupt
*/
{
/* If there is output to do and everything is ready, do it (local
device is
 * known ready).
 * Notify TTY when the buffer goes empty.
 */
/* si el estado en que estamos contiene los bits activos de nuestro
dispositivo listo, hay algo en la cola y el software externo está
listo, entonces escribimos en el THR un byte del buffer */
if (rs->ostate >= (ODEVREADY | OQUEUED | OSWREADY)) {
   /* Bit test allows ORAW and requires the others. */
#if (MACHINE == IBM_PC)
   out_byte(rs->xmit_port, *rs->otail);
#else /* MACHINE == ATARI */
   MFP->mf_udr = *rs->otail;
#endif
/* Actualizamos los punteros */
   if (++rs->otail == bufend(rs->obuf)) rs->otail = rs->obuf;
/* restamos 1 al contador de bytes del buffer */
```

```
  if (--rs->ocount == 0) {
    rs->ostate ^= (ODONE | OQUEUED);  /* ODONE on, OQUEUED off */
/* si hemos terminado de escribir lo comunicamos a la TTY */
    rs->tty->tty_events = 1;
    force_timeout();
  } else
/* si hemos llegado al umbral por debajo llamamos a la TTY para que
siga enviando bytes al buffer de salida*/
  if (rs->ocount == RS_OLOWWATER) {  /* running low? */
    rs->tty->tty_events = 1;
    force_timeout();
  }
  }}
```

modem_int (rs)

Si ha ocurrido una desconexión se notifica a la TTY y se actualiza el estado.

Si el dispositivo (modem) no está preparado actualiza el estado.

Si el dispositivo (modem) está preparado actualiza el estado y además intenta transmitir.

```
PRIVATE void modem_int(rs)
register rs232_t *rs;                    /* line with modem interrupt */
{
/* Get possibly new device-ready status, and clear ODEVREADY if necessary.
 * If the device just became ready, restart output.
 */

#if (MACHINE == ATARI)
  /* Set active edge interrupt so that next change causes a new interrupt
*/
  MFP->mf_aer = (MFP->mf_aer | (IO_SCTS|IO_SDCD)) ^
      (MFP->mf_gpip & (IO_SCTS|IO_SDCD));
#endif
/* si ha ocurrido una desconexión (no hay portadora)*/
  if (devhup(rs)) {
/* Actualiza el estado */
   rs->ostate |= ODEVHUP;
/* llamamos a la TTY */
   rs->tty->tty_events = 1;
   force_timeout();
   }
if (!devready(rs)) /* si el modem no está listo */
   rs->ostate &= ~ODEVREADY; /* Actualiza el estado */
else if (!(rs->ostate & ODEVREADY)) {/* y si el modem está listo */
   rs->ostate |= ODEVREADY; /* Actualiza el estado */
   if (txready(rs)) out_int(rs);
/* si transmisor preparado intentamos transmitir */
   }
}
```

line_int (rs)

Incrementa los contadores de los errores ocurridos.

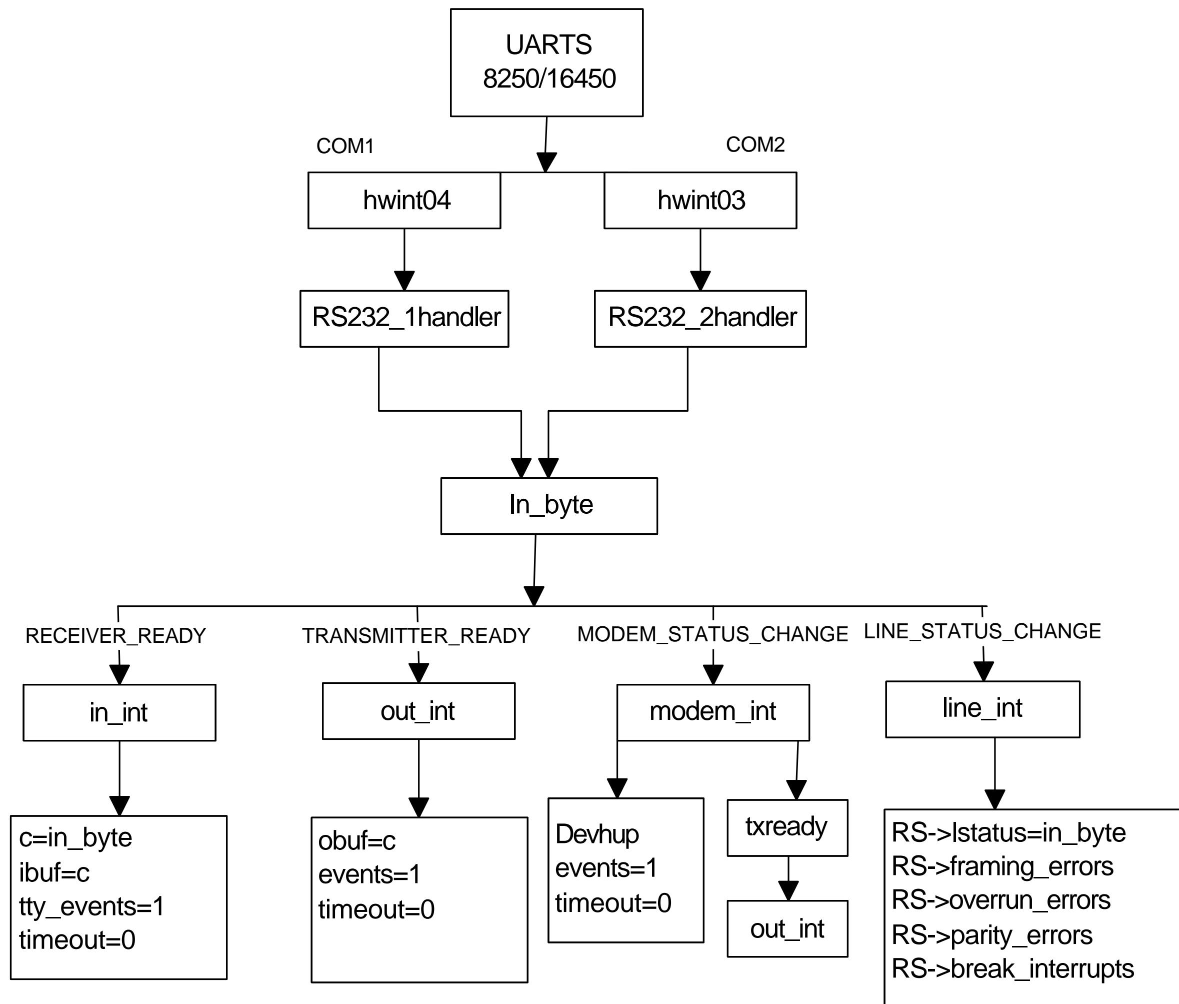
```
PRIVATE void line_int(rs)
register rs232_t *rs;                /* line with line status
interrupt */
{
/* Check for and record errors. */

/* Lee el registro de estado de la linea (LSR) */
#if (MACHINE == IBM_PC)
  rs->lstatus = in_byte(rs->line_status_port);
#else /* MACHINE == ATARI */
  rs->lstatus = MFP->mf_rsr;
  MFP->mf_rsr &= R_ENA;
  rs->pad = MFP->mf_udr;    /* discard char in case of LS_OVERRUN_ERR
*/
#endif /* MACHINE == ATARI */
  if (rs->lstatus & LS_FRAMING_ERR) ++rs->framing_errors;
  if (rs->lstatus & LS_OVERRUN_ERR) ++rs->overrun_errors;
  if (rs->lstatus & LS_PARITY_ERR) ++rs->parity_errors;
  if (rs->lstatus & LS_BREAK_INTERRUPT) ++rs->break_interrupts;
}
```

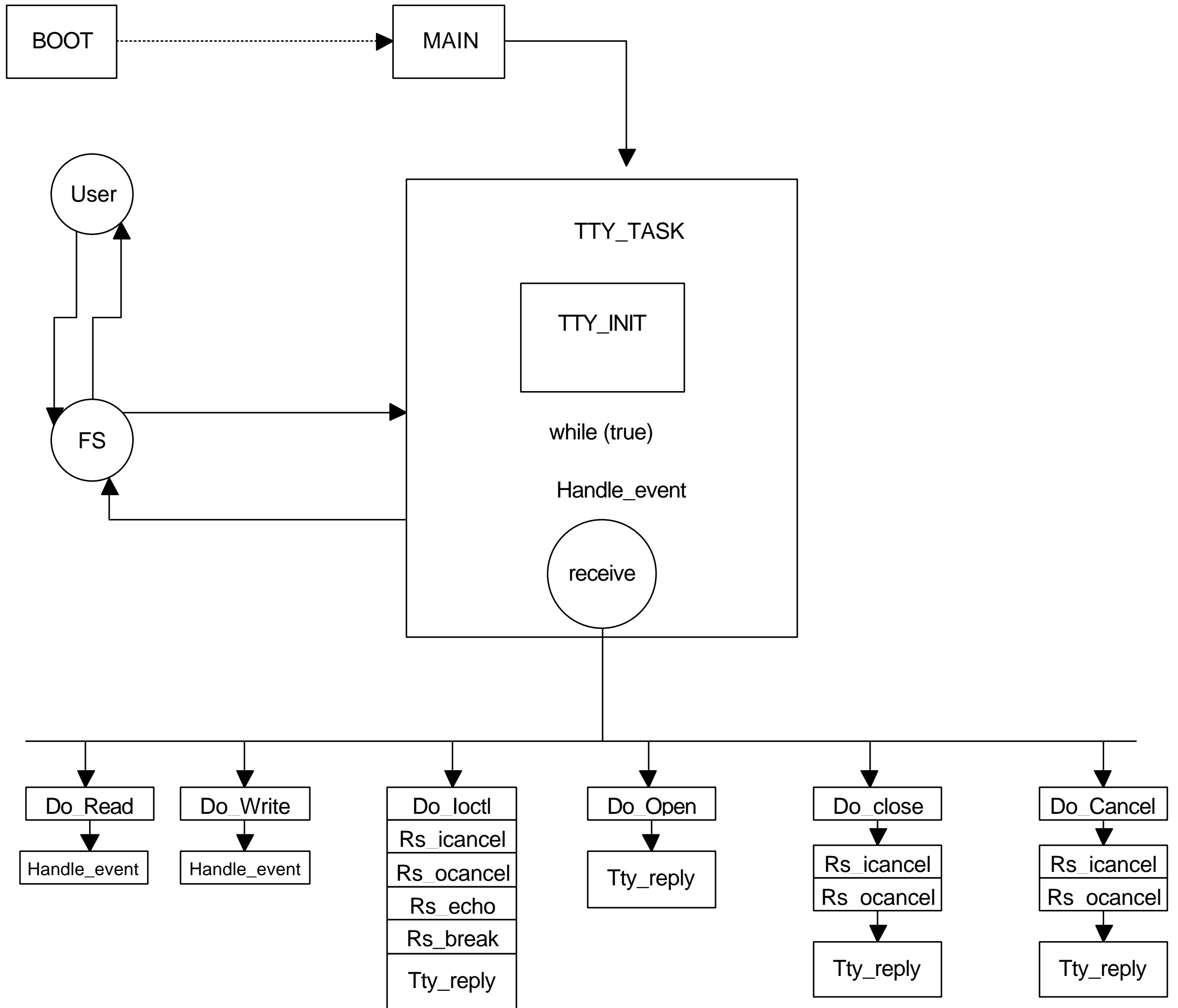
Preguntas

1. Utilidad y objetivos del manejador del puerto serie RS-232.
2. Explicar la estructura principal del manejador (rs232_t).
3. Explicar el manejador de interrupciones de la rs-232 (Rs232_handler)

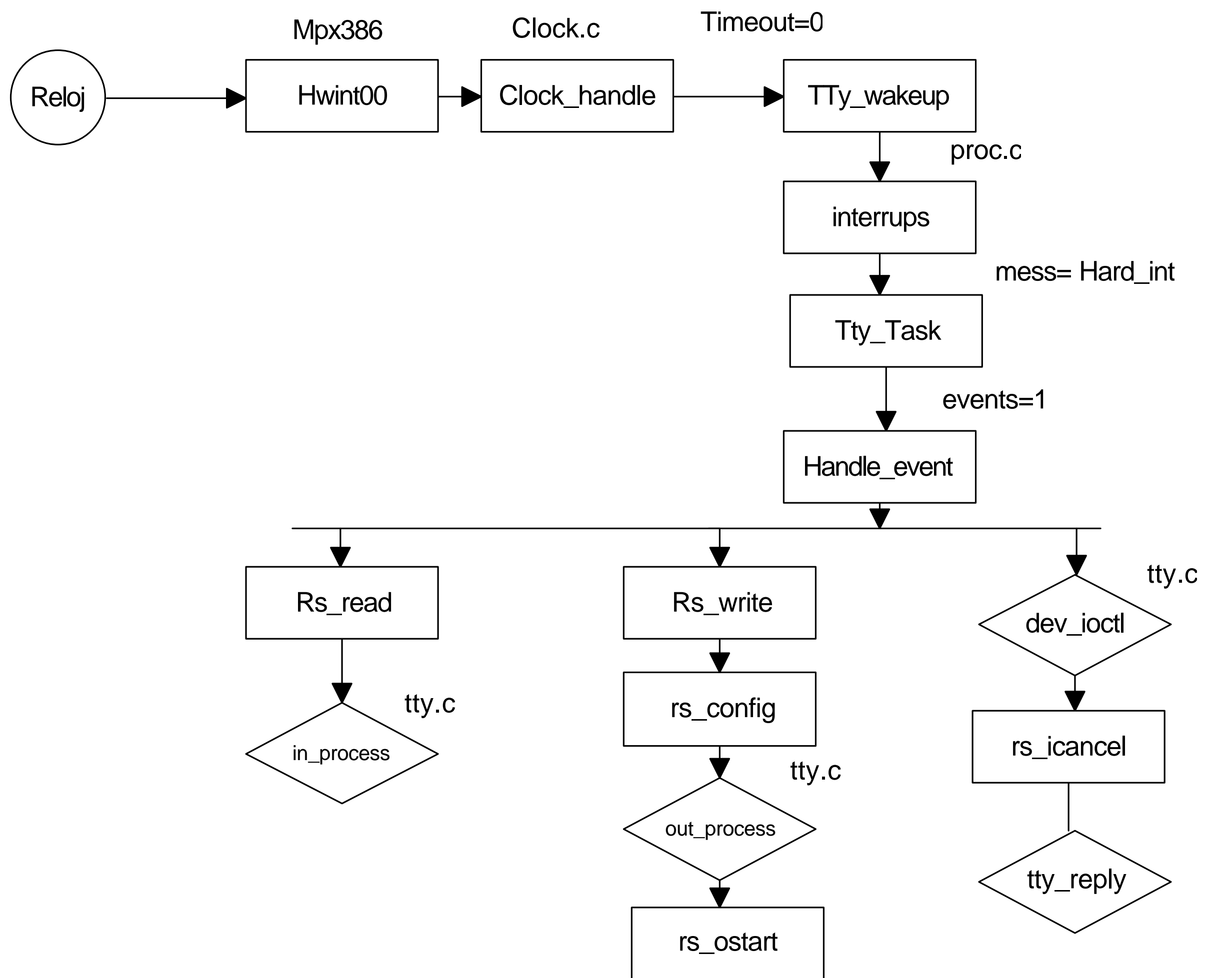
Interrupción de la UART



Petición de usuario



BOOT
MAIN
User
TTY_TASK
TTY_INIT
FS
while (true)
Handle_event
receive
Do_Read
Handle_event
Do_Write
Handle_event
Do_Ioctl
Rs_icancel
Rs_ocancel
Rs_echo
Rs_break
Tty_reply
Do_Open
Tty_reply
Do_close
Rs_icancel
Rs_ocancel
Tty_reply
Do_Cancel
Rs_icancel
Rs_ocancel
Tty_reply


Mpx386
Clock.c
Timeout=0
Reloj
Hwint00
Clock_handle
TTy_wakeup
proc.c
interrups
mess= Hard_int
Tty_Task
events=1
Handle_event
Rs_read
Rs_write
tty.c
dev_ioctl
tty.c
in_process
rs_config
rs_icancel
tty.c
out_process
tty_reply
rs_ostart