

23 jun 02 19:46 **serie.c** Página 1/6

```

/*****
/* SERIE.C      (c) Juan Gonzalez.   Junio-2002
/* -----
/* Licencia GPL
/* Libreria para comunicaciones serie. Especialmente adaptada para trabajar
/* con la tarjeta CT6811.
*****/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <fcntl.h>
#include <sys/time.h>      /* Implementacion de timeouts */
#include <linux/serial.h>

#include "serie.h"

static int fd;          /* Descriptor del puerto serie abierto */
static int npuerto;
extern int errno;
static struct termios oldtermios;

static int nerror;
static char error_serie[10][80]={
    {"Error al hacer TIOCGSERIAL"}, /* 1 */
    {"Error al hacer TIOCSSERIAL"}, /* 2 */
    {"Error al hacer TCGEST"},      /* 3 */
    {"Error al hacer TCSETSF"},     /* 4 */
    {"-----"},                  /* 5 */
    {"Error en TIOCMSET"},          /* 6 */
    {"Error en TIOCMSET"},          /* 7 */
    {"Error en TCGETATTR"},         /* 8 */
    {"Error en TCFLUSH"},           /* 9 */
    {"Error en TCSETATTR"},         /* 10 */
};

char *getserial_error()
/*****
/* Devolver la cadena de error del ultimo error producido
*****/
{
    if (nerror==0) return NULL;
    if (nerror==5) return (char *)strerror(errno); /* Error en open */
    else return (char *)&error_serie[nerror-1]; /* Otro error */
}

int setup_serial()
/*****
/* Configurar la estructura termios para trabajar con el puerto serie.
/* Configurar para 8 bits de datos y sin paridad.
*****/
{
    struct termios newtermios;

    /* Leer atributos del puerto */
    if (tcgetattr(fd,&oldtermios)==-1) {
        nerror=8;
        return 0;
    }

```

23 jun 02 19:46 **serie.c** Página 2/6

```

newtermios.c_cflag= CBAUD | CS8 | CLOCAL | CREAD;
newtermios.c_iflag=IGNPAR;
newtermios.c_oflag=0;
newtermios.c_lflag=0;
newtermios.c_cc[VMIN]=1;
newtermios.c_cc[VTIME]=0;

/* Vaciar buffer de entrada */
if (tcflush(fd,TCIFLUSH)==-1) {
    nerror=9;
    return 0;
}

/* Vaciar buffer de salida */
if (tcflush(fd,TCOFLUSH)==-1) {
    nerror=9;
    return 0;
}

if (tcsetattr(fd,TCSANOW,&newtermios)==-1) {
    nerror=10;
    return 0;
}
return 1;
}

int baudios(int vel)
/*****
/* Configurar la velocidad de transmision
*****/
{
    speed_t baud;
    struct termios mytermios;

    /* Leer atributos del puerto */
    if (tcgetattr(fd,&mytermios)==-1) {
        nerror=8;
        return 0;
    }

    switch(vel) {
        case 1200 : baud=B1200; break;
        case 7680 : baud=B38400; break;
        case 9600 : baud=B9600; break;
        default  : baud=B9600; break;
    }

    cfsetospeed(&mytermios,baud);
    cfsetispeed(&mytermios,baud);

    if (tcsetattr(fd,TCSANOW,&mytermios)==-1) {
        nerror=10;
        return 0;
    }
    return 1;
}

int vaciar_buffer_tx()
/*****
/* Vaciar el buffer de transmision
*****/

```

23 jun 02 19:46	serie.c	Página 3/6
<pre> { if (tcflush(fd,TCOFLUSH)==-1) { nerror=5; return 0; } return 1; } int vaciar_buffer_rx() /* ***** */ /* Vaciar el buffer de recepcion */ /* ***** */ { if (tcflush(fd,TCIFLUSH)==-1) { nerror=5; return 0; } return 1; } int abrir_puerto_serie(int puerto) /* ***** */ /* Abrir el puerto serie para trabajar con el. */ /* El parametro puerto indica el puerto serie que se quiere */ /* abrir: */ /* 0 --> COM1 (ttyS0) */ /* 1 --&; COM2 (ttyS1) */ /* Por defecto se abre a la velocidad de 7680 */ /* ***** */ { char disp[15]; switch(puerto){ case 0 :strcpy(disp, "/dev/ttyS0"); break; /* COM1 */ case 1 :strcpy(disp, "/dev/ttyS1"); break; /* COM2 */ default :strcpy(disp, "/dev/ttyS1"); } npuerto=puerto; fd = open(disp,O_RDWR O_NOCTTY O_NONBLOCK); /* Abrir puerto serie */ if (fd==-1) { nerror=5; return 0; } if (setup_serial()==0) return 0; /* Configurar puerto serie */ return 1; } int enviar_break() /* ***** */ /* Enviar una senal de BREAK */ /* ***** */ { if (tcsendbreak(fd,1)==-1) { nerror=5; return 0; } return 1; } int enviar_bloque(char *cad, int tam) </pre>		

23 jun 02 19:46	serie.c	Página 4/6
<pre> /* ***** */ /* Enviar una cadena por el puerto serie */ /* ***** */ { if (write(fd,cad,tam)==-1) { nerror=5; return 0; } return 1; } int enviar_car(char car) /* ***** */ /* Enviar un caracter por el puerto serie */ /* ***** */ { if (write(fd,&car,1)==-1) { nerror=5; return 0; } return 1; } int car_waiting() /* ***** */ /* Detectar si hay caracteres esperando a ser leidos. Se devuelve */ /* 0 si no hay caracteres esperando. En cualquier otro caso se */ /* devuelve el numero de caracteres esperando en el buffer de */ /* recepcion. */ /* ***** */ { int n; if (ioctl(fd,TIOCIHQ,&n)==0) { return n; } return 0; } int bufftx_waiting() /* ***** */ /* Detectar si hay caracteres pendientes de ser enviados. La */ /* Funcion devuelve el numero de caracteres pendientes de */ /* transmision. Devuelve 0 si se han enviado todos los datos */ /* ***** */ { int n; if (ioctl(fd,TIOCOHQ,&n)==0) { return n; } return 0; } char leer_car() /* ***** */ /* Leer un caracter del buffer de recepcion. Si no hay ninguno */ /* se espera hasta que venga */ /* ***** */ { char c; while (!car_waiting()); read(fd,&c,1); } </pre>		

23 jun 02 19:46

serie.c

Página 5/6

```

}
return c;
}

char leer_car_plazo(int plazo, int *timeout)
/*****
/* Leer un caracter del buffer de recepcion. Si el caracter no se ha
/* recibido dentro del plazo establecido se devuelve un valor disinto
/* de cero en timeout. En caso de llegar un caracter dentro del plazo
/* indicado se devuelve el caracter y se devuelve 0 en timeout
/*
/* El plazo de tiempo esta especificado en microsegundos
*****/
{
    fd_set leer;
    struct timeval tout;
    int n;
    char c;

    tout.tv_sec=0;
    tout.tv_usec=plazo;

    FD_ZERO(&leer);
    FD_SET(fd,&leer);

    n=select(fd+2,&leer,NULL,NULL,&tout);
    if (n==0) {
        *timeout=1;
        return 0;
    }
    *timeout=0;
    read(fd,&c,1);
    return c;
}

int wait_break(int plazo)
/*****
/* Esperar la senal de break durante el plazo especificado. El plazo
/* tiene unidades de microsegundos. La funcion devuelve:
/* 0 Timeout
/* 1 Ha llegado la senal de break
*****/
{
    char c;
    int timeout;

    c=leer_car_plazo(plazo,&timeout);
    if (timeout) return 0;
    if (c!=0) return 0;
    else return 1;
}

int getdtr()
/*****
/* Devolver el estado del DTR
*****/
{
    int arg;

    ioctl(fd,TIOCMGET,&arg);
    /* return arg&TIOCM_DTR; */
    if ((arg&TIOCM_DTR)==0) return 0;
    else return 1;
}

```

23 jun 02 19:46

serie.c

Página 6/6

```

}

int dtr_on()
/*****
/* Activar el DTR
*****/
{
    int flag=0;
    int status;

    flag=TIOCM_DTR;
    status = ioctl(fd,TIOCMBSIS,&flag);
    if (status !=0) {
        nerror=6;
        return 0;
    }
    return 1;
}

int dtr_off()
/*****
/* Desactivar el DTR
*****/
{
    int flag=0;
    int status;

    flag=TIOCM_DTR;
    status = ioctl(fd,TIOCMBSIC,&flag);
    if (status !=0) {
        nerror=7;
        return 0;
    }
    return 1;
}

int getserial_fd()
/*****
/* Devolver el descriptor del fichero del puerto serie abierto
*****/
{
    return fd;
}

void cerrar_puerto_serie()
/*****
/* Cerrar el puerto serie
*****/
{
    /* Reestablecer antiguos atributos */
    if (tcsetattr(fd,TCSANOW,&oldtermios)==-1) {
        printf ("Error en TCSETATTR\n");
        exit(1);
    }

    close (fd);
}

```