CÓMO Programar el puerto serie en Linux

por Peter H. Baumann, Peter.Baumann@dlr.de
traducción de Pedro Pablo Fábrega pfabrega@arrakis.es
v0.3, 14 Junio 1997

Este documento describe cómo programar comunicaciones con dispositivos sobre puerto serie en una máquina Linux.

Índice General

1 Introducción

Este es el COMO Programar el puerto serie en Linux. Todo sobre cómo programar comunicaciones con otros dispositivos/ordenadores sobre una línea serie, bajo Linux. Explicaremos diferentes técnicas: E/S Canónica (sólo se transmiten/reciben líneas completas), E/S asíncrona, y espera para una entrada de origen múltiple.

Este documento no describe cómo configurar un puerto serie, ya que esto ha sido descrito por Greg Hankins en el Serial-HOWTO[1]

Tengo que hacer notar encarecidamente que no soy un experto en este campo, pero he tenido problemas con un proyecto que necesitaba tales comunicaciones. Los ejemplos de código añadidos aquí se derivaron del código de *miniterm* disponible en la guía de programadores del *Linux Documentation Project*:

(`ftp://sunsite.unc.edu/pub/Linux/docs/LDP/programmers-guide/` y espejos) en el directorio de ejemplos. Si alguien tiene algún comentario, con gusto lo incorporaré a este documento (ver sección 1.3).

Todos los ejemplos fueron comprobados usando un núcleo Linux 2.0.29 sobre un i386.

1.1 Copyright

1.2 Versiones futuras de este Documento

Las nuevas versiones de COMO Programar el puerto serie en Linux estarán disponibles en:
`ftp://sunsite.unc.edu:/pub/Linux/docs/HOWTO/Serial-Programming-HOWTO` y sus espejos. Hay otros formatos, como versiones PostScript y DVI en el directorio `other-formats`.

CÓMO Programar el puerto serie en Linux también está disponible en
`http://sunsite.unc.edu/LDP/HOWTO/Serial-Programming-HOWTO.html` y será enviado a
`comp.os.linux.answers` mensualmente.

1.3 Revisión

Por favor, mándeme cualesquiera corrección, pregunta, comentario, sugerencia o material adicional. Me gustaría mejorar este HOWTO-COMO. Dígame exactamente qué es lo que no entiende, o qué debería estar más claro. Me puede encontrar en `Peter.Baumann@dlr.de` vía email. Por favor, incluya el número de versión del CÓMO Programar el puerto serie en Linux cuando escriba. Esta es la versión 0.3.

[1] Disponible en castellano como *Serie-COMO*.

2 Comenzando

2.1 Depuración

La mejor forma de depurar su código es configurar otra máquina Linux y conectar los dos ordenadores mediante un cable null-módem.

Use `miniterm`, disponible en el LDP *Programmers Guide*:
(`ftp://sunsite.unc.edu/pub/Linux/docs/LDP/programmers-guide/`
en el directorio de ejemplos) para transmitir caracteres a su máquina Linux. Miniterm se puede compilar con mucha facilidad y transmitirá todas las entradas en bruto del teclado por el puerto serie.

Sólo las sentencias *define* (`#define MODEMDEVICE "/dev/ttyS0"`) tienen que ser comprobadas. Ponga `ttyS0` para COM1, `ttyS1` para COM2, etc.. Es esencial para comprobar que *todos* los caracteres se transmiten en bruto (sin un procesamiento de salida) por la línea. Para comprobar su conexión, inicie `miniterm` en ambos ordemadores y teclee algo. Los caracteres introducidos en un ordenador deberían aparecer en el otro y viceversa. La entrada no tendrá eco en la pantalla del ordenador en el que escribamos.

Para hacer un cable null-modem tiene que cruzar las líneas TxD (transmit) y RxD (receive). Para una descripción del cable vea el Serie-COMO.

También es posible ejecutar estas comprobaciones con un sólo ordenador, si tiene un puerto serie no utilizado. Puede ejecutar dos miniterm en sendas consolas virtuales. Si libera un puerto serie desconectando el ratón, recuerde redirigir `/dev/mouse`, si existe. Si usa una tarjeta multipuerto serie, esté seguro de configurarla correctamente. Yo tenía la mía mal configurada, y todo funcionaba bien mientras hacía las comprobaciones en un sólo ordenador. Cuando lo conecté a otro, el puerto empezó a perder caracteres. La ejecución de dos programas en un ordenador nunca es completamente asíncrona.

2.2 Configuración del Puerto

Los dispositivos `/dev/ttyS*` tienen como misión conectar terminales a su linux, y están configurados para este uso al arrancar. Hay que tener esto presente cuando se programen comunicaciones con un dispositivo. Por ejemplo, los puertos están configurados para escribir en pantalla cada carácter enviado desde el dispositivo, que normalmente tiene que ser cambiado para la transmisión de datos.

Todos los parámetros se pueden configurar fácilmente con un programa. La configuración se guarda en una estructura `struct termios`, que está definida en <`asm/termbits.h`>:

```
#define NCCS 19
struct termios {
      tcflag_t c_iflag;         /* parametros de modo entrada */
      tcflag_t c_oflag;         /* parametros de modo salida */
      tcflag_t c_cflag;         /* parametros de modo control */
      tcflag_t c_lflag;         /* parametros de modo local */
      cc_t c_line;              /* disciplina de la linea */
      cc_t c_cc[NCCS];          /* caracteres de control */
};
```

Este archivo también incluye todas las definiciones de parámetros. Los parámetros de modo entrada de `c_iflag` manejan todos los procesos de entrada, lo cual significa que los caracteres enviados desde el dispositivo pueden ser procesados antes de ser leídos con `read`.

De forma similar `c_oflag` maneja los procesos de salida. `c_cflag` contiene la configuración del puerto, como la velocidad en baudios, bits por carácter, bits de parada, etc... Los parámetros de modo local se guardan en `c_lflag`. Determinan si el carácter tiene eco, señales enviadas al programa, etc...

Finalmente la tabla `c_cc` define el carácter de control para el fin de fichero, parada, etc... Los valores por defecto de los caracteres de control están definidos en <`asm/termios.h`>. Los parámetros están descritos en la página del manual `termios(3)`.

La estructura `termios` contiene los elementos `c_line`. Estos elementos no se mencionan ni las páginas del manual para *termios* de Linux ni en las páginas de manual de Solaris 2.5. ¿Podría alguien arrojar alguna luz sobre esto? ¿No debería estar incluido en la estructura `termio`?

2.3 Conceptos de entrada para dispositivos serie

Hay tres diferentes conceptos de entrada que queremos presentar. El concepto apropiado se tiene que escoger para la aplicación a la que lo queremos destinar. Siempre que sea posible no haga un bucle para leer un sólo carácter a fin de obtener una cadena completa. Cuando he hecho esto, he perdido caracteres, mientras que un `read` para toda la cadena no mostró errores.

2.3.1 Proceso de Entrada Canónico

Es el modo de proceso normal para terminales, pero puede ser útil también para comunicaciones con otros dispositivos. Toda la entrada es procesada en unidades de líneas, lo que significa que un `read` sólo devolverá una línea completa de entrada. Una línea está, por defecto, finalizada con un `NL`(ASCII `LF`), y fin de fichero, o un carácter fin de línea. Un `CR` (el fin de línea por defecto de DOS/Windows) no terminará una línea con la configuración por defecto.

El proceso de entrada canónica puede, también, manejar los caracteres borrado, borrado de palabra, reimprimir carácter, traducir `CR` a `NL`, etc..

2.3.2 Proceso de Entrada No Canónico

El *Proceso de Entrada No Canónico* manejará un conjunto fijo de caracteres por lectura, y permite un carácter temporizador. Este modo se debería usar si su aplicación siempre lee un número fijo de caracteres, o si el dispositivo conectado envía ráfagas de caracteres.

2.3.3 Entrada Asíncrona

Los dos modos descritos anteriomente se pueden usar en modos síncrono y asíncrono. El modo síncrono viene por defecto, donde la sentencia `read` se bloquará hasta que la lectura esté completa. En modo asíncrono la sentencia `read` devolverá inmediatamente y enviará una señal al programa llamador cuando esté completa. Esta señal puede ser recibida por un manejador de señales.

2.3.4 Espera de Entradas Origen Múltiple

No es un modo diferente de entrada, pero puede ser útil si está manejando dispositivos múltiples. En mi aplicación manejaba entradas sobre un *socket* TCP/IP y entradas sobre una conexión serie de otro ordenador de forma casi simultánea. El programa ejemplo dado abajo esperará una entrada de dos orígenes distintos. Si la entrada de una fuente está disponible, entonces será procesada, y el programa esperará otra entrada nueva.

La aproximación presentada abajo parece más bien compleja, pero es importante tener en cuenta que Linux es un sistema operativo multiproceso. La llamada al sistema `select` no carga la CPU mientras espera una entrada, mientras que un bucle hasta que hay una una entrada disponible ralentizaría demasiado el resto de procesos que se ejecuten a la misma vez.

3 Programas Ejemplo

Todos los ejemplos provienen de `miniterm.c`. El buffer está limitado a 255 caracteres, como la longitud máxima de cadena para el proceso de entrada canónica. (<`linux/limits.h`> o <`posix1_lim.h`>).

Vea los comentarios que hay en el código para una explicación del uso de los diferentes modos de entrada. Espero que el código sea comprensible. El ejemplo de entrada canónica está mejor comentado, el resto de los ejemplos están comentados sólo donde difieren del ejemplo de entrada canónica para remarcar las diferencias.

Las descripciones no son completas, por eso le invito a experimentar con los ejemplos para obtener mejores soluciones para su aplicación.

¡No olvide dar los permisos apropiados a los puertos serie:

```
chmod a+rw /dev/ttyS1
```

3.1 Proceso de Entrada Canónico

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

/* la tasa de baudios esta definida en  <asm/termbits.h>, que esta
   incluida <termios.h> */

#define BAUDRATE B38400

/* cambie esta definicion por el puerto correcto  */
#define MODEMDEVICE "/dev/ttyS1"

#define _POSIX_SOURCE 1 /* fuentes cumple POSIX  */

#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

main()
{
   int fd,c, res;
   struct termios oldtio,newtio;
   char buf[255];

/*
      Abre el dispositivo modem para lectura y escritura y no como controlador
      tty porque no queremos que nos mate si el ruido de la linea envia
      un CTRL-C.
*/

   fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
   if (fd <0) {  perror(MODEMDEVICE); exit(-1);  }

   tcgetattr(fd,&oldtio); /* almacenamos la configuracion actual del puerto */
```

```
  bzero(newtio, sizeof(newtio)); /* limpiamos struct para recibir los
                                    nuevos parametros del puerto */

/*
    BAUDRATE: Fija la tasa bps. Podria tambien usar cfsetispeed y cfsetospeed.
    CRTSCTS : control de flujo de salida por hardware (usado solo si el cable
    tiene todas las lineas necesarias Vea sect. 7 de Serial-HOWTO)
    CS8     : 8n1 (8bit,no paridad,1 bit de parada)
    CLOCAL  : conexion local, sin control de modem
    CREAD   : activa recepcion de caracteres
*/

  newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

/*
    IGNPAR  : ignora los bytes con error de paridad
    ICRNL   : mapea CR a NL (en otro caso una entrada CR del otro ordenador
    no terminaria la entrada) en otro caso hace un dispositivo en bruto
    (sin otro proceso de entrada)
*/

  newtio.c_iflag = IGNPAR | ICRNL;

/*
    Salida en bruto.
*/
  newtio.c_oflag = 0;

/*
    ICANON  : activa entrada canonica
    desactiva todas las funcionalidades del eco, y no envia segnales al
    programa
    llamador
*/

  newtio.c_lflag = ICANON;

/*
    inicializa todos los caracteres de control
    los valores por defecto se pueden encontrar en /usr/include/termios.h,
    y vienen dadas en los comentarios, pero no los necesitamos aqui
*/

  newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
  newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
  newtio.c_cc[VERASE]   = 0;     /* del */
  newtio.c_cc[VKILL]    = 0;     /* @ */
  newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
  newtio.c_cc[VTIME]    = 0;     /* temporizador entre caracter, no usado */
  newtio.c_cc[VMIN]     = 1;     /* bloqu.lectura hasta llegada de caracter. 1 */
  newtio.c_cc[VSWTC]    = 0;     /* '\0' */
  newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
  newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
  newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
  newtio.c_cc[VEOL]     = 0;     /* '\0' */
  newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
  newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
```

```
        newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
        newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
        newtio.c_cc[VEOL2]    = 0;     /* '\0' */

      /*
           ahora limpiamos la linea del modem y activamos la configuracion del
           puerto
      */

        tcflush(fd, TCIFLUSH);
        tcsetattr(fd,TCSANOW,&newtio);

      /*
           configuracion del terminal realizada, ahora manejamos las entradas.
           En este ejemplo, al introducir una  'z' al inicio de linea terminara el
           programa.
      */

        while (STOP==FALSE) {     /* bucle hasta condicion de terminar */

      /*
        bloque de ejecucion de programa hasta que llega un caracter de fin de
        linea, incluso si llegan mas de 255 caracteres.
        Si el numero de caracteres leidos es menor que el numero de caracteres
        disponibles, las siguientes lecturas devolveran los caracteres restantes.
        'res' tomara el valor del numero actual de caracteres leidos.
      */

                           res = read(fd,buf,255);
                           buf[res]=0;             /* envio de fin de cadena, a fin de poder usar printf */
                           printf(":%s:%d\n", buf, res);
                           if (buf[0]=='z') STOP=TRUE;
                         }

      /* restaura la anterior configuracion del puerto  */

        tcsetattr(fd,TCSANOW,&oldtio);
      }
```

3.2 Proceso de Entrada NO Canónico

En el modo de proceso de entrada no canónico, la entrada no está ensamblada en líneas y el procesamiento de la entrada (`erase, kill, delete`, etc.) no ocurre. Dos parámetros controlan el comportamiento de este modo: `c_cc[VTIME]` fija el temporizador de carácter, y fija el número mínimo de caracteres a recibir antes de satisfacer la lectura.

Si MIN $>$ 0 y TIME $=$ 0, MIN fija el número de caracteres a recibir antes de que la lectura esté realizada. Como TIME es cero, el temporizador no se usa.

Si MIN $=$ 0 y TIME $>$ 0, TIME indica un tiempo de espera. La lectura se realizará si es leído un sólo carácter, o si se excede TIME (t =TIME *0.1 s). Si TIME se excede, no se devuelve ningún carácter.

Si MIN $>$ 0 y TIME $>$ 0, TIME indica un temporizador entre caracteres. La lectura se realizará si se reciben MIN caracteres o el tiempo entre dos caracteres excede TIME. El temporizador se reinicia cada vez que se recibe un carácter y sólo se hace activo una vez que se ha recibido el primer carácter.

Si MIN $=$ 0 y TIME $=$ 0, la lectura se realizará inmediatamente. Devolverá el número de caracteres disponibles en el momento, o el número de caracteres solicitados. De acuerdo con Antonino (ver contribuciones), podría poner un `fcntl(fd, F_SETFL, FNDELAY);` antes de leer para obtener el mismo resultado.

Modificando `newtio.c_cc[VTIME]` y `newtio.c_cc[VMIN]` se pueden comprobar todos los modos descritos arriba.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* fuentes cumple POSIX */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

main()
{
   int fd,c, res;
   struct termios oldtio,newtio;
   char buf[255];

   fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
   if (fd <0) {  perror(MODEMDEVICE); exit(-1);  }

   tcgetattr(fd,&oldtio); /* salva configuracion actual del puerto  */

   bzero(newtio, sizeof(newtio));
   newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
   newtio.c_iflag = IGNPAR;
   newtio.c_oflag = 0;

/* pone el modo entrada (no-canonico, sin eco,...) */

   newtio.c_lflag = 0;

   newtio.c_cc[VTIME]    = 0;   /* temporizador entre caracter, no usado */
   newtio.c_cc[VMIN]     = 5;   /* bloquea lectura hasta recibir 5 chars  */

   tcflush(fd, TCIFLUSH);
   tcsetattr(fd,TCSANOW,&newtio);


   while (STOP==FALSE) {                            /* bucle para entrada */
                         res = read(fd,buf,255);   /* devuelve tras introducir 5 */
                         buf[res]=0;               /* asi podemos printf... */
                         printf(":%s:%d\n", buf, res);
                         if (buf[0]=='z') STOP=TRUE;
                      }
   tcsetattr(fd,TCSANOW,&oldtio);
}
```

3.3 Entrada Asíncrona

```
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* fuentes cumple POSIX  */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

void signal_handler_IO (int status);   /* definicion del manejador de segnal */
int wait_flag=TRUE;                    /* TRUE mientras no segnal recibida */

main()
{
   int fd,c, res;
   struct termios oldtio,newtio;
   struct sigaction saio;           /* definicion de accion de segnal  */
   char buf[255];

/* abre el dispositivo en modo no bloqueo (read volvera inmediatamente) */

      fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
   if (fd <0) {  perror(MODEMDEVICE); exit(-1);  }

/* instala el manejador de segnal antes de hacer asincrono el dispositivo */

   saio.sa_handler = signal_handler_IO;
   saio.sa_mask = 0;
   saio.sa_flags = 0;
   saio.sa_restorer = NULL;
   sigaction(SIGIO,&saio,NULL);

/* permite al proceso recibir SIGIO */

      fcntl(fd, F_SETOWN, getpid());

/* Hace el descriptor de archivo asincrono (la pagina del manual dice solo
      O_APPEND y O_NONBLOCK, funcionara con  F_SETFL...) */

   fcntl(fd, F_SETFL, FASYNC);
   tcgetattr(fd,&oldtio); /* salvamos conf. actual del puerto */

/*
      fija la nueva configuracion del puerto para procesos de entrada canonica
*/

   newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
   newtio.c_iflag = IGNPAR | ICRNL;
   newtio.c_oflag = 0;
```

```
   newtio.c_lflag = ICANON;
   newtio.c_cc[VMIN]=1;
   newtio.c_cc[VTIME]=0;
   tcflush(fd, TCIFLUSH);
   tcsetattr(fd,TCSANOW,&newtio);

/* bucle de espera para entrada. Normalmente se haria algo util aqui  */

   while (STOP==FALSE) {
                        printf(".\n");usleep(100000);

/*
tras recibir SIGIO, wait_flag = FALSE, la entrada esta disponible y puede ser leida
 */

                if (wait_flag==FALSE) {
                                   res = read(fd,buf,255);
                                   buf[res]=0;
                                   printf(":%s:%d\n", buf, res);
                                   if (res==1) STOP=TRUE; /* para el bucle si solo entra un CR */
                                   wait_flag = TRUE;      /* espera una nueva entrada */
                                }
                        }
/* restaura la configuracion original del puerto  */
      tcsetattr(fd,TCSANOW,&oldtio);
}


/***************************************************************************
* manipulacion de segnales. pone wait_flag a FALSE, para indicar al bucle  *
* anterior que los caracteres han sido recibidos                           *
***************************************************************************/

void signal_handler_IO (int status)
{
   printf("recibida segnal SIGIO.\n");
   wait_flag = FALSE;
}
```

3.4 Espera de Entradas de Origen Múltiple.

Esta sección está al mínimo. Sólo intenta ser un indicación, y por tanto el ejemplo de código es pequeño. Esto no sólo funcionará con puertos serie, sino que también lo hará con cualquier conjunto de descriptores de archivo.

La llamada `select` y las macros asociadas usan un `fd_set`. Esto es una tabla de bits, que tiene una entrada de bit para cada número de descriptor de archivo válido. `select` aceptará un `fd_set` con los bits fijados para los descriptores de archivos relevantes y devuelve un `fd_set`, en el cual los bits para el descriptor del archivo están fijados donde ocurre una entrada, salida o excepción. Todas la manipulaciones de `fd_set` se llevan a cabo mediante las macros proporcionadas. Ver también la página del manual `select(2)`.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

main()
```

```
{
   int    fd1, fd2;  /* origenes de entrada  1 y 2 */
   fd_set readfs;    /* descriptor de archivo */
   int    maxfd;     /* mixmum file desciptor used */
   int    loop=1;    /* bucle mientras TRUE */

/*
   open_input_source abre un dispositivo, fija el puerto correctamente
   y devuelve un descriptor de archivo
*/

   fd1 = open_input_source("/dev/ttyS1");   /* COM2 */
   if (fd1<0) exit(0);
   fd2 = open_input_source("/dev/ttyS2");   /* COM3 */
   if (fd2<0) exit(0);
   maxfd = MAX (fd1, fd2)+1;  /* entrada maxima de bits (fd) a probar */

/* bucle para entrada */
      while (loop) {
                      FD_SET(fd1, &readfs);  /* comprobacion origen 1  */
                      FD_SET(fd2, &readfs);  /* comprobacion origen 2 */

/* bloqueo hasta que la entrada esta disponible  */
                         select(maxfd, &readfs, NULL, NULL, NULL);
                      if (FD_ISSET(fd1))         /* entrada de origen 1 esta disponible */
                         handle_input_from_source1();
                      if (FD_ISSET(fd2))         /* entrada de origen 2 esta disponible */
                         handle_input_from_source2();
                   }

}
```

El ejemplo dado bloquea indefinidamente hasta que una entrada de una de las fuentes está disponible. Si necesita un temporizador para la entrada, sólo sustituya la llamada `select` por:

```
int res;
struct timeval Timeout;

/* fija el valor del temporizador en el bucle de entrada  */
Timeout.tv_usec = 0;  /* milisegundos */
Timeout.tv_sec  = 1;  /* segundos */
res = select(maxfd, &readfs, NULL, NULL, &Timeout);
if (res==0)
/* numero de descriptores de archivo con input = 0, temporizador sobrepasado */
```

Este ejemplo concluye el tiempo de espera tras un segundo. Si este tiempo transcurre, `select` devolverá 0, pero tenga cuidado porque `Timeout` se decrementa por el tiempo actualmente esperado para la entrada por `select`. Si el valor de retardo es cero, `select` volverá inmediatamente.

4 Otras fuentes de Información

- El *Linux Serie-COMO* describe cómo configurar un puerto serie y contiene información sobre hardware.
- *Serial Programming Guide for POSIX Compliant Operating Systems*, por Michael Sweet.

- La página del manual `termios(3)` describe todos los parámetros de la estructura `termios`.

5 Contribuciones

Como se mencionó en la introducción, no soy un experto en este campo, pero he tenido mis propios problemas, y encontré la solución con la ayuda de otras personas. Gracias por la ayuda de Mr. Strudthoff de *European Transonic Windtunnel*, Cologne, Michael Carter, `mcarter@rocke.electro.swri.edu`, y Peter Waltenberg, `p.waltenberg@karaka.chch.cri.nz`

Antonino Ianella, `antonino@usa.net` escribió el *Serial-Port-Programming Mini HOWTO*, a la misma vez que yo preparaba este documento. Greg Hankins me pidió que incorporara el Mini-Howto de Antonino en este documento.

La estructura de este documento y el formateo SGML provienen del Serial-HOWTO de Greg Hankins.

5.1 Traducción

Este documento ha sido traducido por

Pedro Pablo Fábrega Martínez, `pfabrega@arrakis.es`

Si encontráis mejoras, añadidos o fallos, de cualquier tipo, indicádmelo para mejorar el documento.

Insultos > /dev/null

6 Anexo: El INSFLUG

El *INSFLUG* forma parte del grupo internacional *Linux Documentation Project*, encargándose de las traducciones al castellano de los Howtos (Comos), así como la producción de documentos originales en aquellos casos en los que no existe análogo en inglés.

En el **INSFLUG** se orienta preferentemente a la traducción de documentos breves, como los *COMOs* y *PUFs* (**P**reguntas de **U**so **F**recuente, las *FAQs*. `:)`), etc.

Diríjase a la sede del INSFLUG para más información al respecto.

En la sede del INSFLUG encontrará siempre las **últimas** versiones de las traducciones: `www.insflug.org`. Asegúrese de comprobar cuál es la última versión disponible en el Insflug antes de bajar un documento de un servidor réplica.

Se proporciona también una lista de los servidores réplica (*mirror*) del Insflug más cercanos a Vd., e información relativa a otros recursos en castellano.

Francisco José Montilla, `pacopepe@insflug.org`.