Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Programación del Puerto Paralelo con Linux I (153660 lectures)

Per **Tomeu Capó i Capó**, ***bigBYTE*** (http://www.museu-tecnologic.org/~tomeu/)

Creado el 23/01/2002 02:38 modificado el 23/01/2002 22:43

En este primer articulo de la serie sobre programación del puerto paralelo, vamos a hablar cómo trabajar con el puerto paralelo a traves de Linux con C.

Introducción

En este primer articulo de la serie sobre programación del puerto paralelo, vamos a hablar cómo trabajar con el puerto paralelo a traves de Linux con C, primero haciendo una pequeña introducción a la programación a nivel del Kernel utilizando primitivas *outb, inb, ...* y más adelante a mas alto nivel utilizando funciones mas conocidas *open, fprintf, ...* a traves del puerto /dev/port.

Este articulo y los siguientes, van orientados a la programación de dispositivos primero a bajo nivel estilo programadores de microcontroladores (AVR8, PIC16Fxx) a traves del PPA. Y también el montaje de targetas de salidas digitales al puerto paralelo para su utilización.

Los primeros pasos

Si alguien ha programado nunca con Turbo C I/O's bajo DOS, recordará las archiconocidas funciones *out, inp, ...* del Turbo C o del ensamblador 8086 las cuales servian para atacar a un puerto de I/O determinado a traves del mapeado de direcciones de memoria. En Linux el parecido de las funciones es claramente exacto, ya que a nivel de kernel tiene implementado estas funciones que directamente llaman a las instrucciones del 8086 (OUTB, INPB). Vease el trozo de la declaración de la funcion *outb*:

```
static __inline void outb (unsigned char value, unsigned short int port)
{
  __asm__ __volatile__ ("outb %b0,%w1": :"a" (value), "Nd" (port));
}
```

Basicamente vamos a utilizar estas funciones para "atacar" los dispositivos enchufados al puero paralelo.

El puerto paralelo

La dirección base llamada BASE, puede ser: 0x378, 0x278, 0x3bc. Dependiendo del puerto a utilizar (/dev/lp0, /dev/lp1, /dev/lp2).

El puerto BASE llamado también como el **Data Port** controla las señales de datos **D0..D7**, cada señal es un bit [0,1] que corresponde a los valores de tensión 0v y +5v. Cuando escribimos en el puerto el valor se queda "congelado" el los pines **D0..D7**

El puerto BASE+1 llamado el **Status Port** el cual solo es de solo lectura, nos permite leer el estado del puerto, este puerto se compone como el anterior de 8 señales:

- 0 RESERVADO
- 1 RESERVADO
- 2 IRQ STATUS
- 3 ERROR

- 4 SLCT
- 5 PE
- 6 ACK
- 7 BUSY

A parte de estos dos hay un tercero llamado **Control port** que es de solo escritura, si se intenta leer nos devuelve lo último que hemos enviado por este puerto. Este puerto tiene la dirección BASE+2 y consta también de 8 bits:

Bit	Nombre	Descripción	Tipo
0	STROBE	Indica a la impresora que la información está completa y que puede imprimir el carácter.	*(Lógica negativa)*
1	AUTO_FDXT	Señal de autoalimentación. Controla la forma de manejar los saltos de linea.	*(Lógica negativa)*
2	INIT	Reinicializa la impresora.	*(Lógica positiva)*
3	SLCT_IN	Señal para indicar a la impresora que se ponga en On-Line.	*(Lógica negativa)*
4	Enable the PPA IRQ	Ocurre cuando en se genera una transaccion ACK a bajo nivel.	*(Lógica positiva)*
5	Extended mode direction	0 = Write, 1 = Read	*(Lógica positiva)*
6 y 7	RESERVADOS		

Todas estas señales se reflejan en el *Pin-out*[1] del conector de 25-pins. Las especificaciones de IBM dicen que los pines 1, 14, 16 y 17 las salidas de control trabajan en colector abierto.

Los primeros ejemplos en C

Ahora vamos a hacer una par de ejemplos de como enviar información al puerto paralelo, utilizando las funciones de bajo nivel para el manejo de I/O. Para utilizar las funciones de manejo de I/O mapeadas se tiene que tener en cuenta que se tiene que dar permiso a un rango de direcciones a las cuales se tiene que acceder, para ello utilizamos la funcion *ioperm*. No nos extrañemos si nos falla el programa y nos da un error de **Permission denied** eso sera por no dar permiso al I/O deseado.

```
ioperm(BASEADDR,rango,activar);
```

El parametro **rango** es para indicar el rango de direcciones a dar permiso y el parametro **activar** 1 = Dar permiso o 0 = Quitar permiso.

```
      .
      .
if(ioperm(BaseAddr,3,1)) {
   perror("Dando permisos");
   exit(1);
}
      .
      .
// Utilizamos el BaseAddr
      .
      .

if(ioperm(BaseAddr,3,0)) {
   perror("Quitando permisos");
   exit(1);
}
```

Una vez dado el permiso al rango de direcciones a utilizar, se podran utilizar funciones del estilo *outb inb*, etc ... Para poder usar estas funciones de bajo nivel se tiene que incluir el *header* sys/io.h y para compilar el programa utilizaremos el parametro -O2.

```
$ gcc -O2 -o mi_prog mi_prog.c
```

Ejemplo 1:

Enviar el carácter FORM-FEED (Salto página) a la impresora.

```
#include <stdio.h>
#include <sys/io.h>
#define direccio_pp 0x378

int main(void)
{

    ioperm(direccio_pp,1,1);  // Damos permiso a 1 direccion

    outb(12,direccio_pp);     // Enviamos el caracter 12 (Page-Feed)

    ioperm(direccio_pp,1,0);  // Quitamos el permiso
}
```

Otra manera de hacerlo a mas alto nivel seria utilizando las funciones de *open,write, etc...*

Ejemplo 2:

Este segundo ejemplo es un poco mas conplejo. El puerto paralelo, a parte de controlar dispositivos estilo: impresoras, ZIP, etc ... Tambien lo podemos utilizar para nuestro proposito utilizando asi sus 8 señales de salidas para por ejemplo controlar 8 relés. En este ejemplo no hemos ido tan lejos y solo controlamos 8 leds haciendo asi un tester del puerto paralelo, para ello utilizamos un CI TTL 74245 de intermediario para los datos de salida **D0..D7** el esquema seria el siguiente[2]:

Un ejemplo típico y vistoso, seria el de la serpiente que va y viene.

```
#include <stdio.h>
#include <sys/io.h>
#define BaseAddr 0x378     /* LPT1 */

int main()
{
     unsigned char p=0x01;
     int sentit=1,i;

     if(ioperm(BaseAddr,3,1)) {
        perror("ioperm")
        exit(1);
     }

     while(1) {


          for(i=0;i<=7;i++) {
           outb(p,BaseAddr);
           if(sentit) {
             if(p1)
```

```
                p >>= 1;     // Desplazamos bits hacia la derecha
             }
             usleep(950);
            }

           if(sentit) {
             sentit=0;
           } else
             sentit=1;

      }

      if(ioperm(BaseAddr,3,0)) {
         perror("ioperm");
         exit(1);
      }
      exit(0);
}
```

Ejemplo 3:

Este ultimo ejemplo ilustra la funcion de como leer el bit de estado **ACK**.

```
#include <stdio.h>
#include <sys/io.h>

int GetLPTAckwl() {
    return(inb(BaseAddr+1) & 0x40); // Mira el bit 6 del byte d'estat
}

int main(void) {
    ioperm(BaseAddr,3,1);

    printf("Esperando respuesta de la impresora");

    while(!GetLPTAckwl())
      printf(".");fflush(stdout);

    printf("\nCaracter recibido sin problemas\n");
    ioperm(BaseAddr,3,0);
    exit(0);
}
```

Todos estos ejemplos utilizando las funciones de bajo nivel, tienen sus ventajar y sus inconvenientes. Una de las ventajas es que trabaja muy rapido con la I/O y el inconveniente es que si utilizamos siempre estas funciones por todos los programas que utilizen I/O y estemos ejecutando un programa que por ejemplo espere caracteres a la entrada haciendo un bucle, esto va a "comer" bastantes recusos de la maquina. Otro inconveninente es que la función *ioperm* solo la puede ejecutar **root**. La solución mejor és utilizar el Parport-API del kernel que son unas funciones que nos permiten atacar el puerto paralelo a alto nivel sin generar "interrupciones" a otros procesos del kernel.

Lista de enlaces de este artículo:

1. http://dmi.uib.es/~tomeu/imatges/25dpinout.jpg
2. http://dmi.uib.es/~tomeu/imatges/ppa_tester.jpg

E-mail del autor: tomeu _ARROBA_ museu-tecnologic.org
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1150