Taller de Tiempo Real para Control Robótico

Ejemplos & Funciones

Dr. Nelson Acosta – nacosta@exa.unicen.edu.ar
INCA / INTIA-Fac. Cs. Exactas - UNCPBA
Tandil - Argentina

Driver: módulos cargables (1)

- El núcleo de Linux usaba una **arquitectura monolítica**, con sistema de archivos, drivers de dispositivos y otros componentes, **enlazados estáticamente** con el núcleo en el **momento del arranque**.
- **Sistema inflexible**. Funcionalidad adicional, entonces (re)compilación completa del núcleo.

  Desarrollo de drivers de dispositivos => ***problemas***
- Inflexibilidad desaparece en la **versión 2.0** del núcleo, carga y descarga **dinámica** de módulos en el núcleo.
- **PERMITE** incorporar código al núcleo del SO sin necesidad de reiniciar la maquina.

Driver: módulos cargables (2)

- **Módulo**: archivo objeto (*.o) con rutinas y datos que son cargados y enlazados con el núcleo.
- Una vez cargado, el código del módulo **reside** en el espacio de direcciones del núcleo y se **ejecuta** enteramente dentro del contexto del núcleo.
- Técnicamente un **módulo puede tener cualquier número de rutinas**, con la única restricción de que las funciones *init_module()* {ejecutada cuando el módulo es cargado} y *cleanup_module()* {ejecutada justamente antes de que el módulo sea descargado del núcleo} deben ser suministradas.

Driver: módulos cargables (3)

- Los archivos "**NOM.o**" pueden obtenerse mediante la compilación de un programa C o como la unión de varios "*.o" (*ld –r* ….).
- Al pasar a formar parte del núcleo, el módulo se ejecuta en **modo supervisor**, teniendo **acceso** a:
  - todas las funciones del núcleo.
  - a las funciones exportadas por módulos cargados previamente.
  - a todo el hardware sin ningún tipo de restricción.
- **Diferencia** con el **código original** del **núcleo**: el módulo se puede **extraer**, liberando todos los recursos utilizados por él.

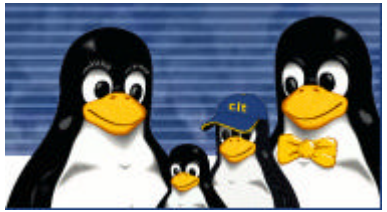
Driver: módulos cargables (4)

- El código de un módulo puede utilizar cualquier función del núcleo, pero este **no ha sido enlazado** con dicho núcleo en tiempo de compilación, las **referencias** a funciones del núcleo no **resueltas**.

Proceso de carga de un módulo:

- Obtener las referencia a funciones ofrecidas por el módulo.
- Incorporar al núcleo las referencias de las funciones ofertadas por el módulo, para que otros módulos las puedan utilizar.
- Insertar el módulo en las zona de memoria del núcleo.
- Invocar a la función *init_module()* del nuevo módulo.

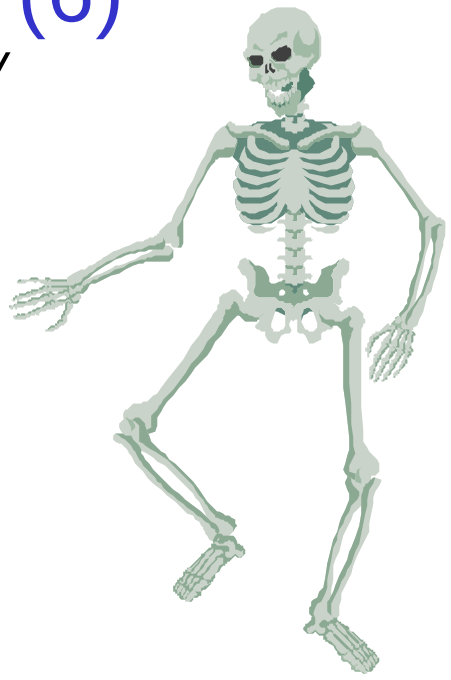
Carga y Descarga de Módulos



Driver: módulos cargables (5)

- Todo estos pasos son realizados por el comando que Linux ofrece para la carga de módulos ***insmod*.**
- Para realizar la descarga del módulo se siguen aproximadamente los mismos pasos pero en orden inverso, invocando a la función *cleanup_molule()* antes de extraer el módulo.
- La descarga se realiza utilizando el comando ***rmmod***, como se verá más adelante.

Driver: módulos cargables (6)



```
/* Código fuente del programa prac2. c */
#ifndef __KERNEL__
#define __KERNEL__
#endif
#ifndef MODULE
#define MODULE
#endif
#define _NO_VERSION_
#include <linux/module.h>
#include <linux/kernel.h>
int salida = 7;
int init_module( void ){
   printk("Modulo cargado.\nSalida=%d.\n",salida);
   return 0;
} /* fin de la función init_module*/
void cleanup_module( void ){
   printk("Modulo bajado.\nSalida=%d \n",salida);
}/* fin de la función cleanup_module*/
```

Driver: módulos cargables (7)

- Para obtener el archivo objeto que será cargado en el núcleo se utilizará el siguiente comando:

```
# gcc -O2 -Wall -DMODULE -D__KERNEL__ -c prac2.c
```

- –D**MODULE** y –D**__KERNEL__** especifican código en forma de módulo cargable.
- –**c** indica que no debe hacer el enlazado (link).

- El núcleo no tiene **salida estándar**, no es posible utilizar *printf()* desde el código de un módulo. *printk()* funciona de manera similar, pero la salida es depositada en un buffer circular de mensajes (***kernel ring buffer***). En cualquier momento puede verse su contenido utilizando el comando *dmesg* o directamente consultando el fichero */proc/kmsg*.

Tabla de Símbolos del núcleo

Driver: módulos cargables (8)

La tabla de símbolos del núcleo

- Cuando de carga un módulo en el núcleo de Linux, cualquier **símbolo global** declarado en el módulo pasa a **formar parte** de la tabla de **símbolos del núcleo**.
- Para ver los símbolos contenidos en dicha tabla puede utilizarse el comando ***ksyms*** o consultar directamente el fichero ***/proc/ksyms***.
- Nuevos módulos pueden **usar** símbolos **exportados** por otros módulos.

Driver: módulos cargables (9)

La tabla de símbolos del núcleo

- Para facilitar la exportación de símbolos, la versión 2.2 y posteriores del núcleo de Linux ofertan las siguientes macros:

`EXPORT_SYMTAB;`

- `Esta macro debe ser definida antes de incluir el archivo <linux/module.h>`

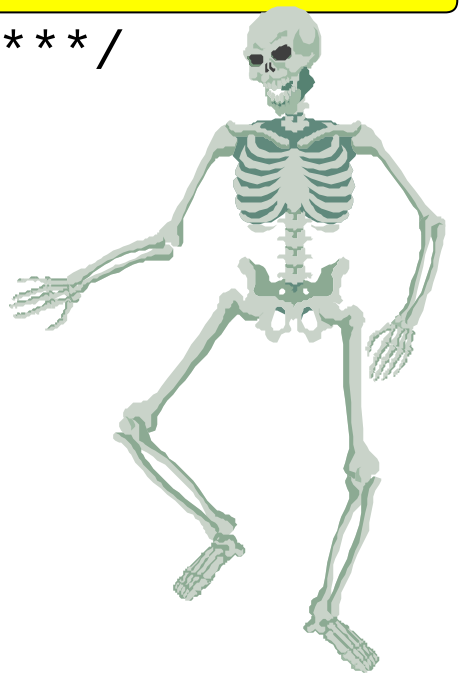
`EXPORT_SYMBOL(name);`

- `Esta macro es usada para exportar el símbolo “name”. No debe ser usada dentro de ninguna función.`

Driver: módulos cargables (10)

```
/*** Código fuente del módulo exportar ***/
#ifndef __KERNEL__
#define __KERNEL__
#endif
#ifndef MODULE
#define MODULE
#endif
#define _NO_VERSION_
#define EXPORT_SYMTAB;
#include <linux/module.h>
#include <linux/kernel.h>
int suma_exportada( int a, int b );
/*llamada a la macro que exporta el símbolo*/
EXPORT_SYMBOL("suma_exportada");
int init_module( void ){   return 0;   }
void cleanup_module( void ){   }
int suma_exportada( int a, int b ){
      return a + b;
}
```

Driver: módulos cargables (11)

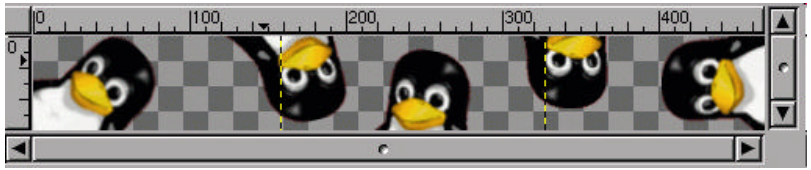


```
/*** Código fuente del módulo importar ***/
#ifndef __KERNEL__
#define __KERNEL__
#endif
#ifndef MODULE
#define MODULE
#endif
#define _NO_VERSION_
#include <linux/module.h>
#include <linux/kernel.h>
int num1 = 4;
int num2 = 6;
extern int suma_exportada( int, int );
int init_module( void ){
   int aux = suma_exportada( num1, num2 );
   printk("Modulo importacion. \n Suma=%d \n", aux );
   return 0;
}
void cleanup_module( void ){   }
```

Comprobación

de

Módulos



Driver: módulos cargables (12)

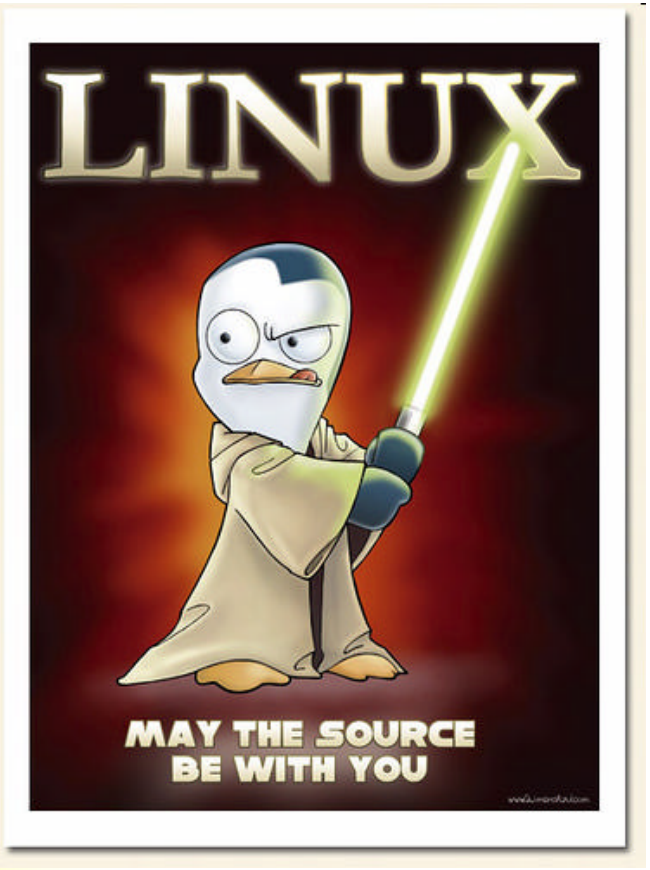
- Para comprobar que el módulo se ha cargado en el núcleo puede usarse el comando ***lsmod*****,** que lista los módulos cargados en el sistema en el momento de realizar la llamada.

```
# lsmod
Module    Size Used by
serial   41940  1  (autoclean)
memstat   1352  0  (unused)
dummy0    7201  1  (autoclean)
```

- La misma información puede obtenerse del archivo ***modules*** del directorio ficticio ***/proc***.

```
# cat /proc/modules
```

Parámetros de Módulos



Driver: módulos cargables (13)

Paso de **parámetros** a los módulos con *insmod*

- Para poder asignar el valor de una variable desde la línea de comandos hay que indicarlo expresamente dentro del módulo con la macro:

```
MODULE_PARM( nombre_variable, "tipo" );
```

- Se añade la sentencia **MODULE_PARM(salida, "i");**, donde la "i" indica que la variable es del tipo int:

```
#insmod parc2.o salida = 3
Módulo prac2 cargado.
Salida=3
```

Importar y exportar:

- El módulo *exportar* tiene que exportar una función que obtenga la suma de dos valores enteros pasados como parámetros.
- Esta función exportada será utilizada dentro de la función *init_module* del módulo *importar* para obtener la suma de dos números enteros elegidos por el módulo.

Driver: módulos cargables (15)

¿ Que pasa si . . .

- Se intenta cargar *importar* sin haber cargado el *exportar*.
- Se carga el *exportar* y el *importar* en ese orden.
- Intenta descargar el módulo *exportar*.
- Se modifica *importar* para que la suma se haga sobre dos números pasados al módulo en el momento de realizar la carga con *insmod*.

Puerto

ParaLelo

Puerto paralelo: el driver … (1)

Manejador para el puerto paralelo:

- El driver permite que cualquier programa de aplicación pueda enviar datos de 8 bits al puerto paralelo y recibir datos de 4 bits a través de las líneas de estado.
- Para poder comprobar que el dato es enviado correctamente se conecta una tarjeta dotada con un **display de 7 segmentos**, conectado directamente a las líneas de datos del puerto paralelo.

Puerto paralelo: el driver … (2)

- Para introducir datos a través de las líneas de estado del puerto, se dispone de cuatro interruptores (para poner un “1” o un “0” por cualquiera de esas líneas).
- El manejador hará accesible dicho hardware de la forma estándar a todas las aplicaciones, a través de un archivo de dispositivo (*/dev/prac3*). Se puede abrir desde cualquier aplicación (*open*), se leer o escribe usando las llamadas al sistema *read* y *write*.

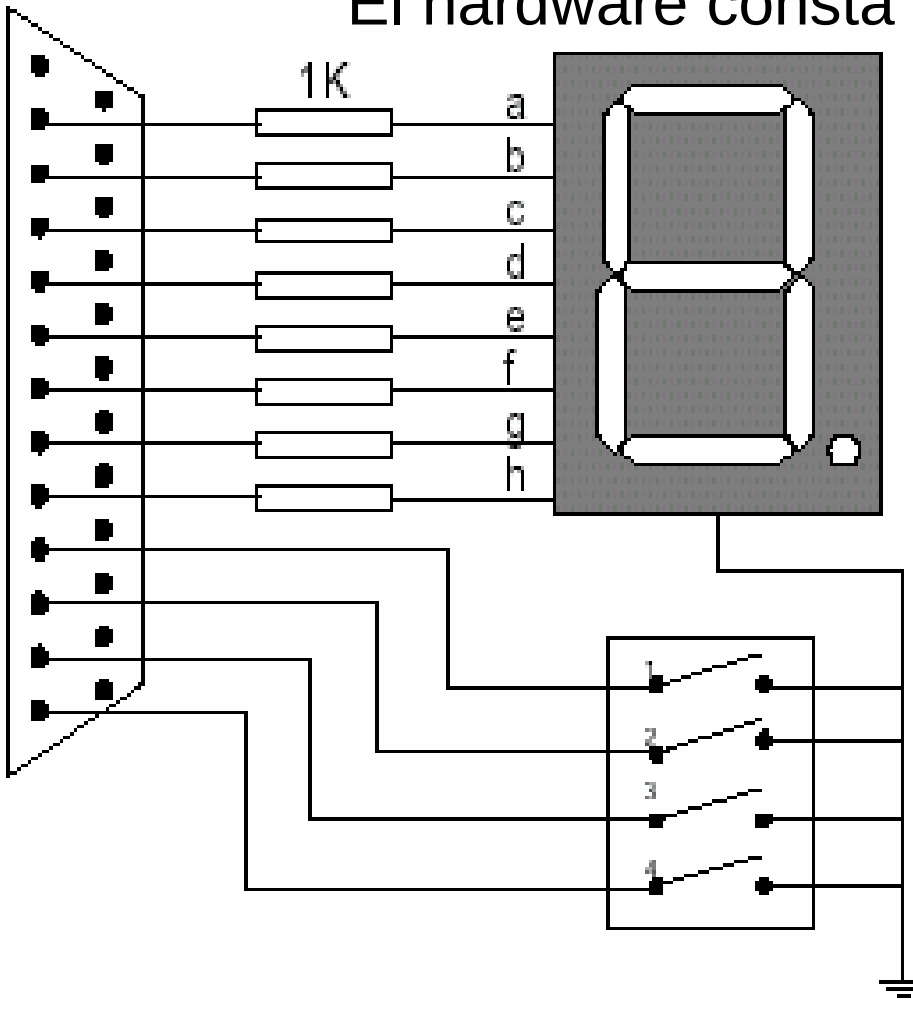
Puerto paralelo: el driver ... (3)

- El núcleo de Linux está formado por múltiples **componentes**, aunque no todos son necesarios para los usuarios en todo momento.
- Al **crearse el núcleo**, Linux solicita al usuario que especifique los componentes que desea incluir en el núcleo.
- El objetivo de esta operación es **insertar únicamente los manejadores de dispositivos que son necesarios** para la configuración de la máquina (**núcleo chico**, más memoria para el usr, arranque más rápido).

Puerto paralelo: el driver … (4)

Descripción del hardware

El hardware consta de un visualizador **LED de 7 segmentos**



conectado directamente a las 8 líneas de datos del puerto paralelo mediante resistencias de 1K y unos interruptores que pueden abrirse o cerrarse, ocasionando la entrada de un valor lógico de “1” o “0” a través de una de las **4** líneas de estado.

Puerto paralelo: el driver … (5)

Descripción del hardware

- Las **conexiones de entrada** al puerto paralelo disponen de una resistencia de **pull-up interna**; y por tanto, para poder introducir un "0" o un "1" a través de una de estas líneas es necesario colocar un interruptor entre la línea en cuestión y masa (0v).
- Cuando el interruptor está cerrado (*on*) se lee un cero; cuando está abierto (*off*) la pata queda al aire exteriormente, pero es puesta a nivel alto por la resistencia de pull-up y por lo que se lee un uno.

Puerto paralelo: el driver … (6)

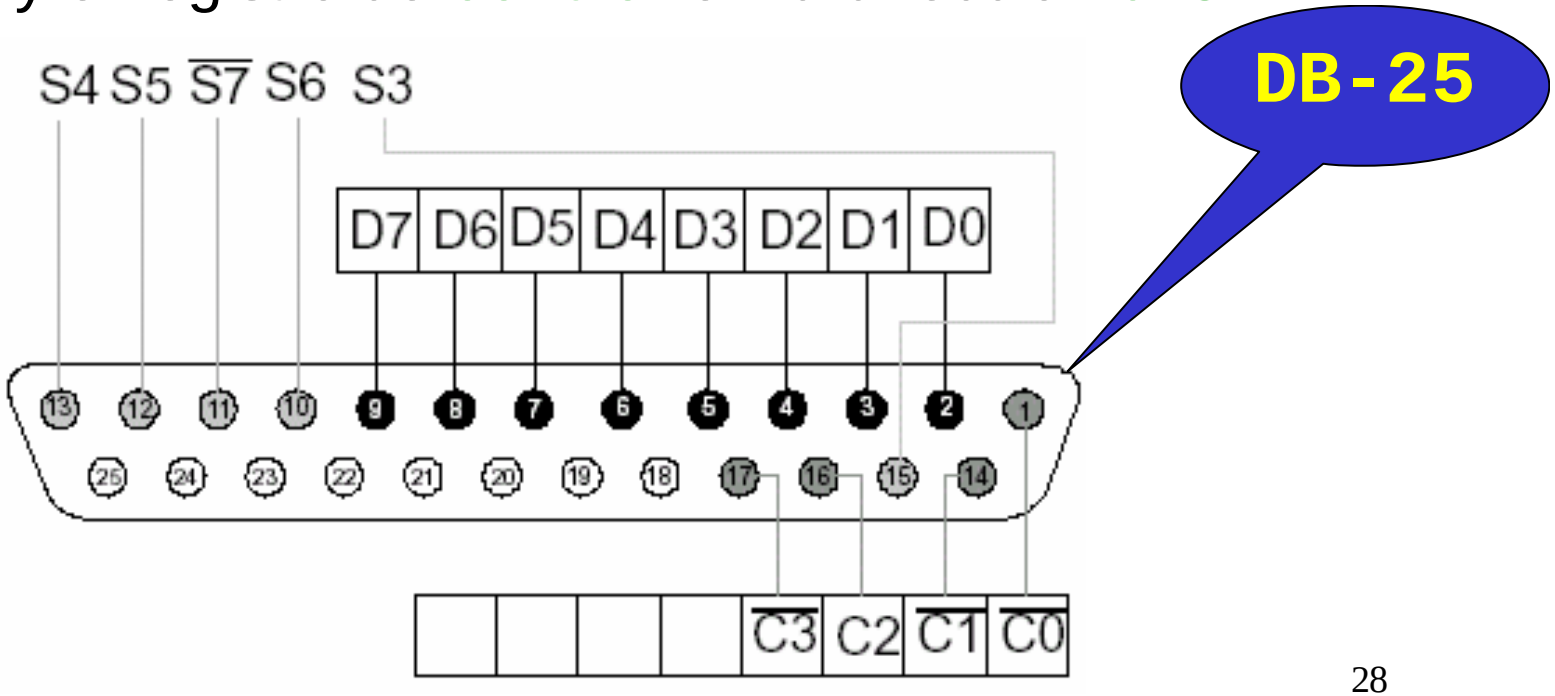
Puerto paraLELO:

- 12 salidas y 5 entradas digitales. **3 registros** de 8 bits, registro de **control**, registro de **estado** y registro de **datos**, consecutivos en el espacio de direcciones de E/S del PC.
- 8 líneas de datos (D0..D7) accesibles en el registro de **DATOS**, 5 entradas (una invertida) accesibles en el registro de **ESTADO** (S3, S4, S5, S6, S7) y cuatro salidas, tres invertidas, accesibles en el registro de **CONTROL** (C0, C1, C2, C3).
- Las 8 patillas restantes del conector **DB-25 hembra** se encuentran conectadas a MASA (0V).
- El **bit 4** del registro de **CONTROL**, C4 permite activar las **interrupciones** por un flanco en la pata **10** del DB25.
- Los otros bits de **ESTADO** y **CONTROL** no se utilizan.

Puerto paralelo: el driver … (6)

Puerto paraLELO:

- La **dirección base** del puerto paralelo suele ser **0x378**, lo que implica que el registro de **datos** se encuentra en la dirección **0x378**, el registro de **estado** en la dirección **0x379** y el registro de **control** en la dirección **0x37A**.



Puerto paralelo: el driver … (7)

Desarrollo del manejador

- **Abstracción** (Linux) de los dispositivos físicos conectados al PC es la del **archivo especial de dispositivo**.
- **Dispositivo sencillo** (puerto paralelo), el manejador únicamente suele implementar las llamadas ***open()***, ***read()*** y ***write()***, ya que con ellas se dispone de toda la funcionalidad disponible en el dispositivo.
- **Funcionalidad más compleja** se logra a través de ***ioctl()***; por ejemplo en la configuración de velocidad de transmisión.

Puerto paralelo: el driver … (8)

Desarrollo del manejador

- Driver sencillo que muestre en el display de 7 segmentos los datos que se escriben en el archivo ***/dev/prac3***, y que cuando se lea el fichero devuelva un valor cuyos bits reflejen el estado de los interruptores.
- El ***major number*** asociado al archivo especial índica al núcleo el manejador que debe atender las llamadas al sistema (*donde se referencia dicho archivo*).
- El ***minor number*** es usado por el manejador para saber sobre cuál de los dispositivos que gestiona va a realizarse la operación.

Puerto paralelo: el driver … (9)

Acceso a las direcciones de I/O del PC

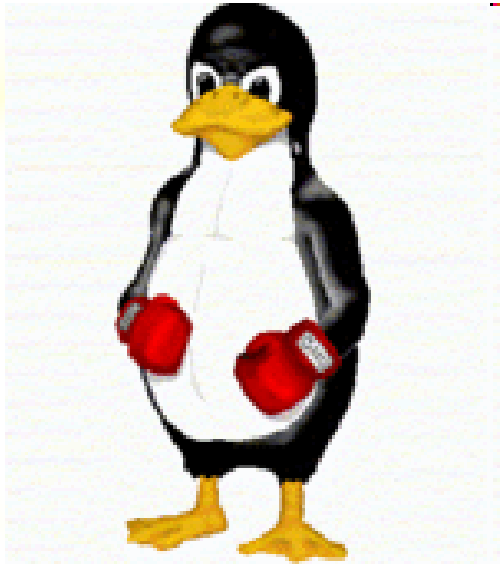
- Acceso a los puertos de E/S desde un programa "C" fuera del núcleo, hay que solicitar al sistema operativo permiso para hacerlo, utilizando la función ***ioperm()***.
- Esta función únicamente puede ser utilizada por el administrador del sistema (**root**).
- Desde un **módulo** no hace falta llamar a esta función, ya que el módulo forma parte del núcleo y se ejecuta con máximos privilegios.

Puerto paralelo: el driver ... (10)

Acceso a las direcciones de I/O del PC

- Funciones de entrada/salida: *inb*(*puerto*), *outb*(*valor*,*puerto*), *inw*(*puerto*) y *outw*(*valor*,*puerto*).
- Desde el núcleo de Linux, el acceso a los puertos de I/O, es regulado por un **mecanismo de reserva de rangos de direcciones**.
- Permite averiguar a un driver, si un determinado rango de direcciones esta siendo usado por otros drivers, ¿**o no**?.

Registro del Puerto ParaLelo



Puerto paralelo: el driver ... (11)

Funciones para evitar colisiones:

- ***int check_region (unsigned int port, unsigned int range):*** Comprueba si un rango (*range*) de puertos a partir del puerto *port* son usados por otro driver (devuelve valor negativo).
- ***void request_region (unsigned int port, unsigned int range, const char *name):*** Reserva un rango (*range*) de puertos a partir de *port* y le asigna el identificador *name*.
- ***void release_region (unsigned int port, unsigned int range):*** Libera un rango de *range* puertos a partir del puerto *port*.

Se invocan desde ***init_module()*** y ***cleanup_module()***.

Se puede averiguar las direcciones reservadas por otros drivers, visualizando el fichero ***/proc/ioports***. ***(#cat /proc/ioports).***

Puerto paralelo: el driver ... (12)

```
register_chrdev( unsigned int major,
                 const char *name,
                 struct file_operations *fops )
```

- el *major number* del archivo especial que queremos atender, un nombre *name*, que identifique al dispositivo y una estructura *fops* que indica al núcleo qué función tiene que invocar para cada tipo de acceso al dispositivo (una para *open()*, otra para *close(),* etc.).
- Registra OK del *major_number*, la función devuelve un valor mayor o igual a 0, sino un código de error negativo.
- No es necesario pasar a la función ***register_chrdev()*** ningún parámetro que tenga relación con el *minor_number* (el núcleo no utiliza el *minor_number*, sólo se lo pasa al manejador cada vez que hay una operación sobre uno de ellos).

Puerto paralelo: el driver ... (13)

- En ***init_module()*** hay que comprobar la disponibilidad del rango de direcciones usado por el dispositivo, en este caso se limita a dos bytes situados a partir de la dirección **0x378**, luego son reservados.

- Cuando se extrae un módulo del núcleo, se debe realizar una operación inversa a la del registrado del *major_number*. Esta operación se lleva a cabo utilizando la función ***unregister_chrdev*** *(unsigned int major, const char * name),* a la que hay que pasarle el *major_number* y el nombre *name*, que fueron usados en el proceso de registrado. También hay que liberar el rango de direcciones de E/S que se reservo durante el proceso de carga.

Puerto paralelo: el driver ... (14)

Funciones de manejo de dispositivo

- La ***struct file_operations *fops*** indica al núcleo que funciones llamar cuando se producen operaciones de apertura (*open*()), cierre (*close*()), lectura (*read*()), escritura (*write*()), etc, sobre el archivo de dispositivo que tiene asociado el *major_number* suministrado también en la llamada a la función *register_chrdev()* referenciada.
- Pueden implementarse sólo algunas funciones.

Puerto paralelo: el driver ... (15)

Estructura apuntada por FOPS:

```
struct file_operations practica3_fops = {
   NULL, /* implementaría practica_lseek */
   practica3_read,
   practica3_write,
   NULL, /* implementaría practica_readdir */
   NULL, /* implementaría practica_select */
   NULL, /* implementaría practica_ioctl */
   NULL, /* implementaría practica_mmap */
   practica3_open,
   NULL,
   practica3_release,
   NULL, /* implementaría practica_fsync */
   NULL, /* implementaría practica_fasync */
};
```

Puerto paralelo: el driver … (16)

int practica3_open(

struct inode *inode, struct file *filp):

- 1) Llamar a macro MOD_INC_USE_COUNT para indicar que el módulo esta siendo usado. Impide que el módulo sea eliminado por error.
- 2) **Retorna 0** si el dispositivo ha sido abierto con éxito. En esta función no se presta atención al *minor number*, ya que nuestro manejador únicamente va a gestionar el primer **puerto paralelo**.
- 3) Si el manejador gestionase varios dispositivos sería necesario consultar el *minor_number*, que se encuentra en el campo *i_rdev* de **inode*.

Puerto paralelo: el driver ... (17)

int practica3_release(struct inode *inode,
struct file *filp):

- La primera operación que debe realizar esta función es decrementar el contador asociado al módulo, llamando a MOD_DEC_USE_COUNT.

Puerto paralelo: el driver … (18)

ssize_t practica3_read (struct file *filp,
char *buf, size_t talla, loff_t *count):

- Utilizar la función *inb()* para leer la dirección de E/S donde se encuentra el registro de estado del puerto paralelo (*bits asociados a los interruptores*), y devolverlos como una cadena de 4 bytes en *buf*. La función devuelve el número de bytes leídos, que en este caso siempre será 4.
- El formato debe reflejar la posición de los interruptores, y un carácter LF (ASCII 12, representado en el lenguaje de programación “C” como \n) al final de la cadena.

Puerto paralelo: el driver … (19)

ssize_t practica3_write (struct file *filp,
char *buf, size_t talla, loff_t *count):

- Usa *outb()* para iluminar los segmentos del visualizador correspondientes al primer carácter de *buf*, devolviendo el número de caracteres escritos, que en este caso siempre será 1.
- Si el carácter no es numérico en ASCII, no se debe iluminar ningún segmento del visualizador.
- El tipo *ssize_t* esta definido como *int*, *size_t* esta definido como *unsigned int* y *loff_t* esta definido como *long*.

Puerto paralelo: el driver ... (20)

- Para robar el funcionamiento tanto del hardware como del driver se puede volcar cualquier archivo sobre el dispositivo con ***#cat datos > /dev/prac3***, aunque la visualización será tan rápida que únicamente se verá el último carácter.
- Una opción más adecuada es la utilización de #***echo –n 5 > /dev/prac3***, que permite visualizar un 5 en el display. Para ver una secuencia de cifras de 0 a 9 es posible usar la siguiente secuencia de ordenes:

```
#for i in 0 1 2 3 4 5 6 7 8 9 ;
do echo –n $i > /dev/prac3; sleep 1; done
```

- Lea los interruptores con: ***# cat /dev/prac3***.

Puerto paralelo: el driver ... (21)

```
#ifndef __KERNEL__
#define __KERNEL__
#endif
#ifndef MODULE
#define MODULE
#endif
#define _NO_VERSION_

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/ioport.h>
#include <asm/io.h>

int practica_major = 55;
int practica_dir_puerto = 0;
int salida = 1;
```

Puerto paralelo: el driver ... (22)

```
int practica3_open (struct inode *inode, struct file *filp){
        MOD_INC_USE_COUNT;
        return 0;
}
int practica3_release (struct inode *inode, struct file *filp)
{       MOD_DEC_USE_COUNT;
        return 0;
}
ssize_t practica3_write (struct file *filp, const char *buf,
                                        size_t talla, loff_t *count){
        unsigned char valor, bits;
        int retval=1;
        valor=*buf;
        if (valor >= '0' && valor <= '9'){
                outb( valor, 0x378 );
                *count = retval;
                return retval;
        }
}
```

Puerto paralelo: el driver ... (23)

```
ssize_t practica3_read (struct file *filp, char *buf,
                                    size_t talla, loff_t *count){
        unsigned char valor;
        char *valores=buf;

        valor= inb( 0x379 );
        valores[0]=(valor & 1000) ? '1' : '0';
        valores[1]=(valor & 0100) ? '1' : '0';
        valores[2]=(valor & 0010) ? '1' : '0';
        valores[3]=(valor & 0001) ? '1' : '0';
        valores[4]='\n';
        *count = 5;
        return 5;
}

void cleanup_module(void) {
        release_region( 0x378, 2 );
        printk("Modulo practica 3 descargado=%d \n", salida);
}
```

Puerto paralelo: el driver ... (24)

```
int init_module ( void ){
        int comprobacion;
        comprobacion = check_region( 0x378, 2 );
        if ( comprobacion  < 0 ) {
                printk("Error en carga prac3=%d \n", salida);
                return -1;
        }
        request_region( 0x378, 2, "prac3“ );
        return 0;
}
struct file_operations practica3_fops = {
        NULL,                   /* implementaría practica_lseek */
        practica3_read,
        practica3_write,
        NULL, NULL, NULL, NULL, /*readdir select ioctl mmap*/
        practica3_open,
        NULL,
        practica3_release,
        NULL, NULL              /* implementaría fsync y fasync */
};
```

Powered by:

Linux™

Esto es todo ...