Programación del teclado bajo linux.

1.- Acceso básico a puertos desde linux.

2.- Programación de módulos en el kernel. Básico.

3.- Avanzado. Codigo de tratamiento de interrupciones. Hacer funcionar. –basado en kernel-Interrupciones. LKM

Periféricos Avanzados.
Depto. Arquitectura de Computadores y Electronica.
Universidad de Almeria

1.- Acceso básico a puertos desde linux.

1.1 Objetivos

· Conocer la forma en que el sistema operativo Linux gestiona los dispositivos conectados al sistema.

· Saber utilizar los diferentes recursos que un driver genérico ofrece para programar un dispositivo de entrada sencillo, el teclado.

1.2 Introducción

La CPU no es el único dispositivo inteligente conectado a un sistema, cada dispositivo físico o periférico posee un controlador hardware. El teclado, el ratón y los puertos serie son controlados por el circuito integrado conocido como SuperIO chip, los discos IDE por un controlador IDE, los dispositivos SCSI por un controlador SCSI, etc.

Cada controlador hardware dispone de registros de control, estado y datos, que son utilizados para inicializar el dispositivo, hacer diagnósticos, intercambiar datos, etc. En lugar de poner el código para manejar un determinado dispositivo dentro de cada aplicación que necesite acceder a el, dicho código es guardado en el núcleo de sistema operativo. El software encargado del manejo de un determinado dispositivo se conoce como manejador/driver de dispositivo. En Linux estos drivers de dispositivos son esencialmente un conjunto de rutinas residentes en memoria y que se ejecutan con los máximos privilegios, ya que forman parte del núcleo, como puede apreciarse en la figura1.

Los drivers de dispositivos pueden ser insertados en el núcleo en forma de módulos, utilizando el comando **insmod**. Esta temática de inserción de módulos de discutirá ampliamente en la segunda parte de la práctica.

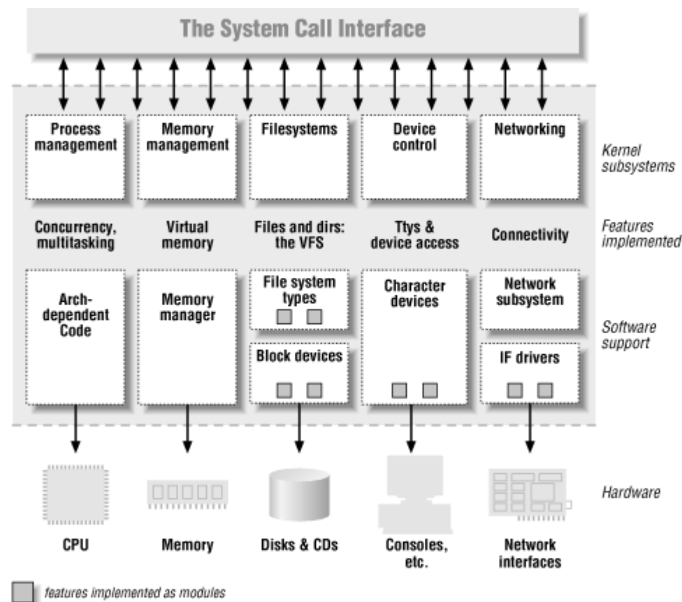

Figura 1. Componentes del núcleo.

Una de las características de Linux es la abstracción que hace del manejo de los dispositivos físicos. Todos los dispositivos son tratados como si fuesen ficheros, pueden ser abiertos, cerrados, leídos y escritos usando las mismas llamadas al sistema que son usadas para manipular ficheros. Cada dispositivo en el sistema es representado por un **fichero de dispositivo (device file),** por ejemplo, el primer disco duro del sistema es representado por */dev/hd*a.

Todos los ficheros de dispositivo presentes en el sistema se encuentran en el directorio */dev.* Se recomienda que el alumno acceda a dicho directorio y ejecute el comando *ls –l.* El resultado obtenido será un listado similar al que se muestra a continuación, aunque mucho más extenso.

```
crw-rw-rw-    1    root root    10, 3, Nov 30 1993 tty0
brw-rw-rw-    1    root disk    2, 0, May 1 1999 fd0
brw-rw-rw-    1    root disk    2, 1, May 1 1999 fd1
brw-rw-rw-    1    root disk    3, 0, May 1 1999 hda
brw-rw-rw-    1    root disk    3, 64, May 1 1999 hdb
crw-rw----    1    root lp      6, 0, May 1 1999 lp0
crw-rw----    1    root lp      6, 1, May 1 1999 lp1
crw-rw-rw-    1    root uucp    4, 64, May 1 1999ttyS0
crw-rw-rw-    1    root uucp    4, 65, May 1 1999ttyS1
```

Cada uno de los ficheros de dispositivo tiene asociado un número que lo identifica y distingue del resto. Este número se conoce como *major numbe*r.

/dev/f0 tiene asociado el *major number* 2, */dev/lp0* tiene asociado el *major number* 6.

Como puede comprobarse, los dispositivos del mismo tipo, como los discos flexibles *fd0* y *fd*1, tienen asociado el mismo *major number* (*2*). Esto quiere decir que deben ser atendidos por el mismo driver. Por el contrario se puede apreciar que muestran *minor mumbers* diferentes (0,1), ya que el *minor number* le indica al driver cuál de los dispositivos que atiende es el que solicita servicio cuando un proceso accede a ese fichero de dispositivo concreto.

Se puede apreciar en el listado anterior, que algunos de los ficheros de dispositivos son identificados por una "c" en la primera columna de la salida obtenida con *ls* –l, indicando que ese fichero especial esta asociado a un dispositivo de carácter. Pueden verse algunos ficheros de dispositivos identificados por la letra "b" en la primera columna, indicando que dicho fichero esta asociado a un dispositivo de bloque. Esto es debido a que Linux distingue entre dos tipos básicos de dispositivos: dispositivos de caracteres y dispositivos de bloques. A continuación se comentan las principales diferencias entre ambos.

1.3 Dispositivos de bloques y caracteres

Dispositivos de carácter: Un dispositivo de carácter puede ser accedido como un fichero normal. El encargado de implementar este proceder es el driver asociado a ese dispositivo. El driver normalmente implementa las llamadas al sistema *open, close, read y writ*e. La consola y el puerto paralelo son dos ejemplos de este tipo de dispositivos, que pueden ser accedidos a través los ficheros especiales */dev/tty* y */dev/l*p.

Dispositivos de bloque: Un dispositivo de bloque es algo que puede albergar un sistema de ficheros, por ejemplo un disco duro. En la mayoría de los sistemas UNIX un dispositivo de bloque puede ser accedido en unidades que son múltiplos del tamaño de bloque, que normalmente es de 1.024 bytes. Linux permite leer y escribir dispositivos de bloque de forma similar a como lo hace para los dispositivos de caracteres, permite transferir cualquier número de bytes a la vez. Tienen acceso aleatorio, lo que implica que cualquier bloque de todos los posibles puede ser accedido en cualquier instante de tiempo. En esta clase se encuentran dispositivo de almacenamiento masivo, como el disco flexible o el disco duro.

En esta primera práctica se va a programar el procesador del teclado, accediendo directamente a sus registros a través de operaciones read() y write().

Se recuerda al alumno que para poder realizar este tipo de operaciones hay que ser root, por lo que hay que tener especial cuidado en no dañar ninguna parte del sistema.

1.4 Actividades a realizar en el laboratorio

El fichero de dispositivo */dev/port* es el que va a permitir el acceso a cualquier puerto de entrada/salida del PC. En está práctica será utilizado para acceder a los puertos 0x60 y 0x64 del controlador de teclado. Como se ha comentado anteriormente Linux trata todos los los dispositivos periféricos como si fuesen ficheros, por lo tanto el primer paso para tener acceso a los puertos de E/S es abrir el fichero */dev/port* **(fd = open("nombrefichero", permisos)).**

Una vez abierto este fichero, el proceso tiene acceso cualquier puerto de E/S de la máquina. Posteriormente hay que indicar sobre que puerto de E/S se quiere realizar la operación de lectura o escritura, utilizando la llamada al sistema **lseek (lseek(fd, puerto,0).**

Por último se realiza sobre ese puerto la operación de lectura o escritura por medio de las llamadas al sistema **read(fd, &dato,1) y write(fd,dato,1);**

A continuación se muestra el esqueleto de un programa en C, que el alumno debe completar para que realice la siguiente tarea:

1) **Mostrar los codigos scan de las teclas pulsadas.**

Unicamente se debe añadir código donde aparece la cadena : **???**?.

Código fuente del que se dispone para la realización de esta parte práctica.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/file.h>


int main(int argc, char **argv)
{
        int i,j,fd;
        unsigned char z;


        if ((fd = open( "?????" , ?????)) < 0)
        {
                perror( "No es posible abrir el fichero /dev/port" );
                exit(1);
        }

        bucle que permite la lectura del puerto de teclado.

                ?????
                ?????
                ?????
                ?????
        return 0;
}
```

1.5 Cómo compilar.

Una vez completado el programa debe compilarse desde la línea de comandos con **gcc –O2 nombre_fuente.c –o nombre_ejecutabl**e. Se recomienda dar al ejecutable el nombre **scan_linux .**

2.- Instalación de módulos en el núcleo.

2.1 Objetivos

Conocer y saber utilizar las herramientas que Linux ofrece para poder realizar de forma dinámica la carga y descarga de módulos.

Entender como la carga dinámica de módulos facilita enormemente la tarea del programador de drivers de dispositivos.

2.2 Introducción

El núcleo de Linux usa una arquitectura monolítica, con sistema de ficheros, drivers de dispositivos y otros componentes, que son enlazados estáticamente con el núcleo en el momento del arranque.

Esta estructura podría dar lugar a un sistema poco flexible, desde el punto de vista de un programador que quiere dotar al sistema de alguna funcionalidad adicional, ya que esto requeriría una recompilación completa del núcleo. Esto puede resultar tedioso cuando se están desarrollando nuevos manejadores de dispositivos. Esta falta de flexibilidad desaparece en la versión 2.0 del núcleo, ya que este incorpora utilidades para la carga y descarga dinámica de módulos en el núcleo. Esta nueva característica permite la incorporación de nuevo código (manejador) al núcleo del sistema operativo **en calient**e, sin necesidad de reiniciar la maquina.

2.3 Qué son los módulos cargables (L.K.M. Loadable Kernel Modules) .

Un módulo es simplemente un fichero objeto (*.o) que contiene rutinas y datos que deben ser cargados o 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()* y *cleanup_module()* deben ser suministradas. La primera es ejecutada cuando el módulo es cargado, y la segunda justamente antes de que el módulo sea descargado del núcleo.

Los ficheros .o pueden ser obtenidos directamente a partir de la compilación de un programa escrito en C, o como la unión de varios ficheros .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 y a todo el hardware de la máquina sin ningún tipo de restricción. La única diferencia con el código original del núcleo, es la posibilidad de extraer el módulo una vez deja de ser útil, liberando de esta forma todos los recursos utilizados por él.

Ya que 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 están resueltas. Por lo tanto el proceso de carga de un módulo ha de seguir los siguientes pasos:

1. Obtener las referencias a funciones ofrecidas por el módulo.

2. Incorporar al núcleo las referencias de las funciones ofertadas por el módulo, para que otros módulos las puedan utilizar.

3. Insertar el módulo en las zonas de memoria correspondiente al núcleo.

4. Por último, invocar a la función *init_module()* del nuevo módulo.

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 ***rmmo***d, como se verá más adelante.

2.4 Creación y utilización de módulos

Un módulo se crea a partir de un programa escrito en C. A continuación se muestra el código fuente de un programa que será utilizado como un módulo.

```
                    /* Código fuente del programa pracLinux2. 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 pracLinux2 cargado en el núcleo. \n salida=%d \n", salida);
   return 0;
 } /* fin de la función init_module*/

void cleanup_module(void)
{
   printk("Modulo pracLinux2 descargado del núcleo. \n salida=%d \n", salida);
}/* fin de la función cleanup_module*/
```

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

gcc -I/usr/src/linux/include/linux –O2 –Wall –DMODULE –D__KERNEL__ -c pracLinux2.c

Las opciones más importantes son –DMODULE y –D__KERNEL__, que especifican que el código será generado en forma de un módulo cargable. La opción –c le indica al compilador que debe detenerse antes de la etapa de enlazado (link). El resultado es el fichero objeto prac2Linux.o .

El núcleo no dispone de salida estándar, por lo que no es posible utilizar *printf()* desde el código de un módulo. El núcleo ofrece una versión de ésta, llamada *print**k**(),* que funciona de manera similar, pero la salida es depositada en un buffer circular de mensajes (kernel ring buffer). En dicho buffer es donde escribe el núcleo todos los mensajes. En cualquier momento puede verse su contenido utilizando el comando *dmesg* o directamente consultando el fichero */proc/kms*g.

Consultar la sintaxis y significado del comando dmesg con : man dmesg

2.4.1 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. Esto es muy importante cuando se desarrollan proyectos muy complejos, ya que permite la realización de módulos básicos que exportan una serie de funciones que pueden ser utilizadas por otros módulos.

3.- Avanzado. Compilar y averiguar la operativa del siguiente codigo fuente, realice una memoria de una pagina donde describa el funcionamiento del siguiente codigo y qué hace.

3.1.- Un poco mas de detalle, manejadores de interrupción:

Una de las tareas más interesantes del kernel es la de comunicarse con el hardware conectado con el pc.

Hay dos tipos de comunicaciones básicas entre los dispositivos hardware y la cpu, cuando la cpu le ordena al hardware alguna tarea y la segunda es cuando el hardware necesita que la cpu haga algo por él. La segunda forma, las ya conocidas interrupciones , es más compleja de implementar por que tiene que "luchar" con el hardware que en ese momento este realizando la petición. Por norma, los dispositivos hardware tienen poca ram por lo que si no se les "hace caso" en un intervalo de tiempo razonable, perderemos la comunicación.

En linux las interrupciones hardware son todas denominadas IRQs. Hay dos tipos de IRQs, sencillas y complejas. Las sencillas ocupan poco tiempo y requieren que el sistema se bloquee para realizar su tarea y no se permitirá ninguna otra interrupción de prioridad igual o menor. Las interrupciones complejas, no bloquean la maquina y no son apropiativas, es decir, permiten que ocurran otras interrupciones pero no del mismo dispositivo. A ser posible, es recomendable realizar los manejadores como interrupciones complejas.

Cuando una interrupción requiere la atencion de la CPU, esta detiene la ejecución de lo que estaba procesando para dedicarse a tratar a la excepcion, la CPU almacena determinados parametros en la pila e invoca al manejador de interrupcion especifico (recordad las practicas del intercambio de VectorViejo - teclado) , esto significa que no es reconmendable incluir determinadas tareas en el manejador, ya que el sistema se encuentra en un estado desconocido (inestable, los registros no contienen informacion fiable). La solución a este problema de inestabilidad está en que el manejador haga lo mas importante y lo más crítico en el menor tiempo posible, normalmente estas tareas son las de leer algo del hardware o mandar algo al hardware, y despues, planificar la gestión de la información para más tarde, es decir, el manejador se compondria de dos mitades, la primera mitad sería la mitad crítica en la que se realizan las tareas criticas en el menor tiempo posible, la segunda mitad, en la que el sistema se haya en un estado de seguridad es cuando se pueden realizar las tareas de gestion de la informacion del manejador, esta segunda parte es llamanda comunmente "la segunda mitad" o B.H. del inglés Bottom Half. Con la segunda mitad se asegura que las tareas se realizarán lo antes posible, aunque hayan sido mandadas al planificador y que ademas se podrán realizar todas las actividades que el kernel te permita por privilegio.

La forma de implementar esto es llamar a "request_irq" para obtener el manejador de interrupcion cuando el IRQ corresponidente sea la que interrumpa al sistema. Hay 16 ORQ en plataformas Intel. Esta función, "request _irq" recibe el nombre del IRQ, el nombre de la función , los flags , un nombre para /proc/interrupts y un parametro para pasarselo al manejador de interrupciones. Los flags pueden incluir SA_SHIRQ para indicar que deseas compartir ese IRQ con otros manejadores de interrupciones (normalmente mas de un dispositivo hardware ocupan la misma IRQ) y SA_INTERRUPT para indicar que es una interrupcion "sencilla" (que tarda poco y bloquea al sistema). Esta función sólo tendrá éxito si no hay ya otro manejador en ese IRQ o incluso de haberlo si permite compartir el IRQ.

Entonces, desde dentro del manejador de interrupciones, nos comunicamos con el hardware y usamos "queue_task_irq" con "tq_immediate" y "mark_bh(BH_IMMEDIATE) para planificar la segunda mitada (BH) . La razon por la que no podemos usar la cola de tareas estandar para planificar a BH es debido a que una interrupcion debido a su naturaleza asíncrona puede aparecer en la mitad de la cola de tareas de cualquiera de los usuarios del sistema. Necesitamos mark_bh porque las versiones antiguas de Linux solo tenian un array de 32 segndas mitades y ahora uno de ellos (BH_IMMEDIATE) se usa para la lista enlazada de segundas mitades de drivers que no tienen una entrada de ese array asignada.

¿Qué hacer?. Hacer funcionar el siguiente codigo y explicar que hace.

```
/* intrpt.c - Un manejador de interrupciones. */

/* Los ficheros de cabeceras necesarios */

/* Estándar en los módulos del núcleo */
#include <linux/kernel.h>   /* Estamos haciendo trabajo del núcleo */
#include <linux/module.h>   /* Específicamente, un módulo */

/* Distribuido con CONFIG_MODVERSIONS */
#if CONFIG_MODVERSIONS==1
#define MODVERSIONS
#include <linux/modversions.h>
#endif

#include <linux/sched.h>
#include <linux/tqueue.h>

/* Queremos una interrupción */
#include <linux/interrupt.h>

#include <asm/io.h>


/* En 2.2.3 /usr/include/linux/version.h se incluye una
 * macro para esto, pero 2.0.35 no lo hace - por lo tanto
 * lo añado aquí si es necesario. */
#ifndef KERNEL_VERSION
#define KERNEL_VERSION(a,b,c) ((a)*65536+(b)*256+(c))
#endif



/* Bottom Half - esto será llamado por el núcleo
 * tan pronto como sea seguro hacer todo lo normalmente
 * permitido por los módulos del núcleo. */

static void got_char(void *scancode)
{
  printk("Código leído %x %s.\n",
    (int) *((char *) scancode) & 0x7F,
    *((char *) scancode) & 0x80 ? "Liberado" : "Presionado");
}


/* Esta función sirve para las interrupciones de teclado. Lee
 * la información relevante desde el teclado y entonces
 * planifica el bottom half para ejecutarse cuando el núcleo
 * lo considere seguro. */
void irq_handler(int irq,
        void *dev_id,
        struct pt_regs *regs)
{
  /* Estas variables son estáticas porque necesitan ser
   * accesibles (a través de punteros) por la rutina bottom
   * half. */
  static unsigned char scancode;
  static struct tq_struct task =
     {NULL, 0, got_char, &scancode};
```

```
  unsigned char status;

  /* Lee el estado del teclado */
  status = inb(0x64);
  scancode = inb(0x60);

  /* Planifica el bottom half para ejecutarse */
#if LINUX_VERSION_CODE > KERNEL_VERSION(2,2,0)
  queue_task(&task, &tq_immediate);
#else
  queue_task_irq(&task, &tq_immediate);
#endif
  mark_bh(IMMEDIATE_BH);
}



/* Inicializa el módulo - registra el manejador de IRQs */
int init_module()
{
  /* Como el manejador de teclado no coexistirá con
   * otro manejador, tal como nosotros, tenemos que deshabilitarlo
   * (liberar su IRQ) antes de hacer algo. Ya que nosotros
   * no conocemos dónde está, no hay forma de reinstalarlo
   * después - por lo tanto la computadora tendrá que ser reiniciada
   * cuando halla sido realizado.
   */
  free_irq(1, NULL);

  /* Petición IRQ 1, la IRQ del teclado, para nuestro
   * irq_handler. */
  return request_irq(
    1,  /* El número de la IRQ del teclado en PCs */
    irq_handler,  /* nuestro manejador */
    SA_SHIRQ,
    /* SA_SHIRQ significa que queremos tener otro
     * manejador en este IRQ.
     *
     * SA_INTERRUPT puede ser usado para
     * manejarla en una interrupción rápida.
     */
    "test_keyboard_irq_handler", NULL);
}


/* Limpieza */
void cleanup_module()
{
  /* Esto está aquí sólo para completar. Es totalmente
   * irrelevante, ya que no tenemos forma de restaurar
   * la interrupción normal de teclado, por lo tanto
   * la computadora está totalmente inservible y tiene que
   * ser reiniciada. */
  free_irq(1, NULL);
}
```