Drivers: Teclado

Christian Ramírez Santana
Fco. José Marzabal Cebreiro

ÍNDICE

- Introducción
- Características del Hardware
- Codificación
- Estructuras de Datos
- Funciones

Introducción

- El teclado es un periférico que consiste en un sistema de teclas, como las de la máquina de escribir, que permite introducir datos a un ordenador o dispositivo electrónico.
- Estructura entera matricial donde las teclas están asociadas a códigos numéricos.
- Básicamente existe dos tipos de organización de las teclas: QWERTY y DVORAK.
- Existen otras variantes poco extendidas como el AZERTY y otros.

Teclado QWERTY

- Diseñado y patentado por Christopher Sholes en 1868.
- Es la distribución de teclado mas común.

Teclado DVORAK

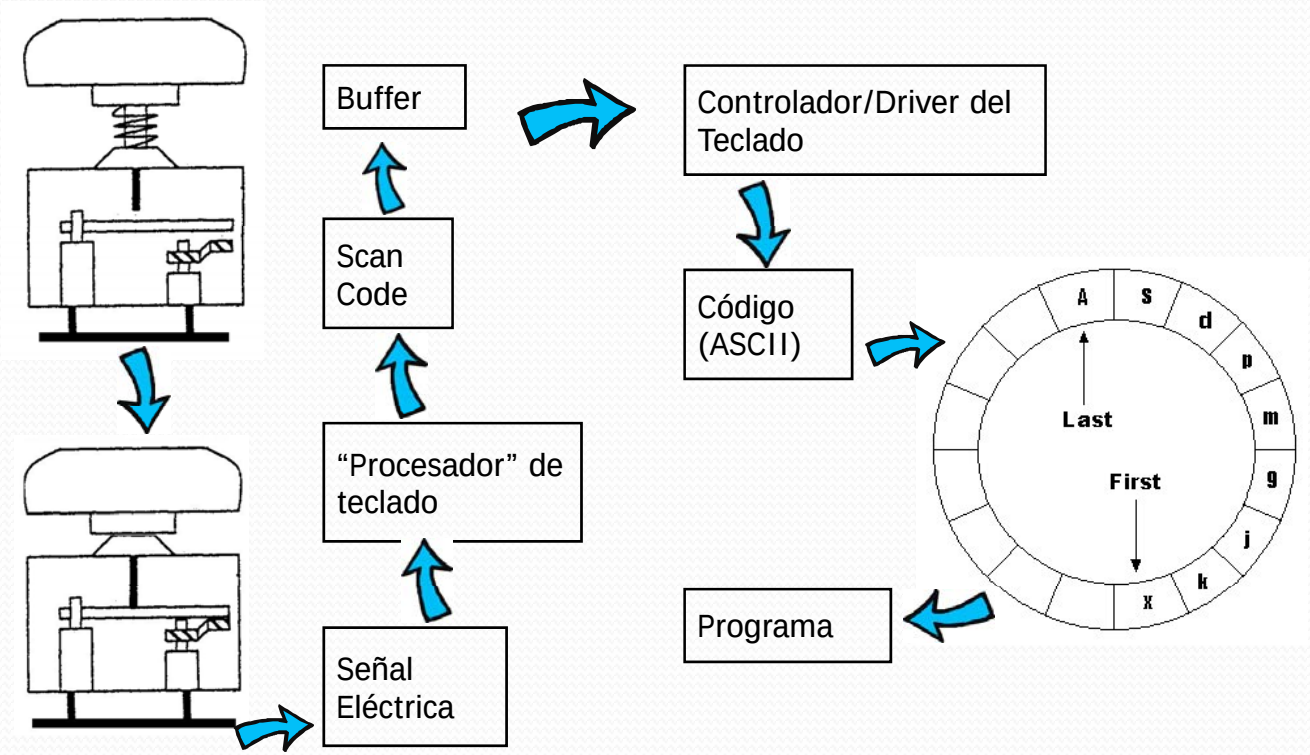
- Diseñado por los doctores August Dvorak y William Dealey en los años 20-30.
- Basado en la frecuencia de letras y fisiología de la mano.

Características del Hardware

- El teclado consta de una matriz de contactos.
- Por cada pulsación o liberación de una tecla el microcontrolador envía un código identificativo llamado **ScanCode.**
- Si el microcontrolador nota que ha cesado la pulsación de una tecla, el nuevo código generado (**BreakCode**) tendrá un valor de pulsación incrementado en 128.
- La interfaz de teclado se ocupa de rastrear continuamente el estado de todas las teclas por si se produce un cambio en cualquiera de ellas.

Características del Hardware

- Al presionarse una tecla, no es añadido directamente al buffer de la tty, es necesario un proceso mas complejo.

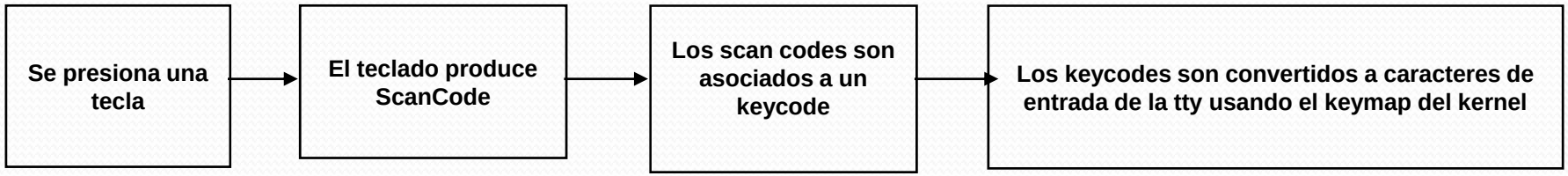


Características Hardware

- Los códigos generados son llamados Códigos de barrido(ScanCode) y son enviados de forma serial a través del conector del teclado(USB o PS/2).
- El código es recibido por el microcontrolador, el cual compara el ScanCode con la Tabla de caracteres.
- Genera una interrupción por hardware y envía los datos al procesador.

Codificación

- El núcleo cuando recibe los ScanCode realiza una traducción a un código interno denominado KeyCode.
- El KeyCode se le pasa al KeyMap para que este devuelva el carácter, secuencia o acción correspondiente al programa de usuario.



Codificación

- ScanCode:
  - MakeCode : se le llama así a la pulsación de una tecla.
  - BreakCode: se le llama así a la liberación de una tecla.
  - Se compone de 8 bits y el MSB diferencia entre pulsación y liberación, por lo que como máximo hay 128 teclas distintas.
  - Al presionar o soltar una tecla se producen secuencias de 1 a 6 bytes que el núcleo tiene que asociar a KeyCode.
  - Cada tecla se asocia a un KeyCode único en un rango de 1 hasta 127.

Codificación

- Los KeyMap son mapas de caracteres empleados para determinar el código de carácter que se le pasa a la aplicación basándose en la tecla que ha sido pulsada y los modificadores activos en ese momento.

- Puede haber varios KeyMap y la razón de estos es que no todos los lenguajes poseen los mismos caracteres y símbolos de puntuación.

Codificación

Fragmento de Linux KeyCode Table

Key	Key Number	Scan Code	Key Code	Keymap			
				Normal	Shift	Alt	Ctrl
Q	16	10 90	16	q 71	Q 51	q 1b 71	11
W	17	11 91	17	w 77	W 57	w 1b 77	17
E	18	12 92	18	e 65	E 45	e 1b 65	05
R	19	13 93	19	r 72	R 52	r 1b 72	12
T	20	14 94	20	t 74	T 54	t 1b 74	14
Y	21	15 95	21	y 79	Y 59	y 1b 79	19

Estructuras de Datos

- **Struct tty_struct:** Estructura de datos genérica del tty (terminal).
- **Struct input_dev:** Estructura que almacena el valor devuelto por el manejador del dispositivo.
- **Struct input_handle:** Estructura del manejador del dispositivo.
- **Struct vc_data:** Estructura de datos referente a la consola.
- **Struct pt_regs:** Estructura que hace referencia a los registros de la CPU

Estructuras de Datos

- **tty_struct**:
  - Definida en linux/tty.h
  - Se usa para efectuar todas las operaciones de entrada realizadas en el núcleo.
  - Indica parámetros como estado de la terminal, tamaño de la ventana, lista de procesos en espera de lectura o escritura…
  - Se soporta en las estructuras tty_driver, tty_ldisc y ktermios que se ven en el tema de la consola.

Estructuras de Datos

- **input_dev**:
  - Definida en linux/input.h
  - Contiene las funciones setkeycode y getkeycode que permiten establecer las relaciones entre los ScanCode y los KeyCode.
  - Los campos keycodemax y keycodesize se usan para definir los tamaños de los KeyCode.
  - Se basa en las estructuras: ff_device, input_handle y device.

Estructuras de Datos

- **input_handle**:
  - Definida en linux/input.h
  - Usa la estructura input_handler que implementa la interfaz para los dispositivos de entrada (funciones como event, connect o disconnect).
  - Contiene la lista de los dispositivos conectados.

Estructuras de Datos

- **vc_data**:
  - Definida en linux/console_struct.h
  - Define la consola asociada a un terminal (tty).
  - La inicialización y las operaciones usan un vector de punteros a vc_data denominado vc_cons[].
  - Cada elemento del vector está vinculado a una tty_struct.
  - Define casi por completo a la consola que representa.

Funciones

- **kbd_connect**: Se llama cuando se conecta un nuevo teclado al ordenador. Rellena la estructura ***input_handle*** del manejador para recibir eventos de entrada.
- **kbd_disconnect**: Se llama cuando se desconecta el teclado. Libera la memoria del manejador asociado.

```
1307  static void kbd_disconnect(struct input_handle *handle)
1308 {
1309       input_close_device(handle);
1310       input_unregister_handle(handle);
1311       kfree(handle);
1312  }
```

Funciones

- **kbd_event**: Cara vez que se pulsa una tecla o una combinación de ellas, se llama a kbd_event e inserta el carácter.

```
1250 static void kbd_event(struct input_handle *handle, unsigned int event_type,
1251                          unsigned int event_code, int value)
1252{
1253        if (event_type == EV_MSC && event_code == MSC_RAW && HW_RAW(handle->dev))
1254                kbd_rawcode(value);
1255        if (event_type == EV_KEY)
1256                kbd_keycode(event_code, value, HW_RAW(handle->dev));
1257        tasklet_schedule(&keyboard_tasklet);
1258        do_poke_blanked_console = 1;
1259        schedule_console_callback();
1260}
```

Funciones

- **kbd_rawcode**: Inserta los caracteres en el buffer en modo RAW, es decir, sin traducir los ScanCode a KeyCode

```
1130 static void kbd_rawcode(unsigned char data)
1131 {
1132        struct vc_data *vc = vc_cons[fg_console].d;
1133        kbd = kbd_table + fg_console;
1134        if (kbd->kbdmode == VC_RAW)
1135               put_queue(vc, data);
1136 }
```

- **kbd_keycode**: Inserta el KeyCode en el buffer.

Funciones

- **getkeycode** : Traducción de ScanCode a KeyCode.

```
170  int getkeycode(unsigned int scancode)
171  {
172          struct input_handle *handle;
173          int keycode;
174          int error = -ENODEV;
175          //Buscamos el primer manejador de teclado que encontremos
176          list_for_each_entry(handle, &kbd_handler.h_list, h_node) {
177                  error = handle->dev->getkeycode(handle->dev, scancode, &keycode);
178                  if (!error)
179                          return keycode;
180          }
181
182          return error;
183  }
```

Funciones

- **setkeycode**: Permite cambiar la asociación entre ScanCode y KeyCode (Para los diferentes símbolos de los idiomas)

```
185 int setkeycode(unsigned int scancode, unsigned int keycode)
186 {
187     struct input_handle *handle;
188     int error = -ENODEV;
189     //Se busca el primer manejador de teclado.
190     list_for_each_entry(handle, &kbd_handler.h_list, h_node) {
191         error = handle->dev->setkeycode(handle->dev, scancode, keycode); //Relaciona SC y KC
192         if (!error)
193             break;
194     }
195     return error;
196 }
```

Funciones

- **put_queue**: Escribe el carácter en un buffer de la terminal.

```
300 static void put_queue(struct vc_data *vc, int ch)
301 {
302         struct tty_struct *tty = vc->vc_tty;
303
304         if (tty) {
305                 tty_insert_flip_char(tty, ch, 0);
306                 con_schedule_flip(tty);
307         }
308 }
```

Funciones

- **puts_queue**: Escribe una secuencia de caracteres en un buffer de la terminal.

```
310 static void puts_queue(struct vc_data *vc, char *cp)
311 {
312         struct tty_struct *tty = vc->vc_tty;
313
314         if (!tty)
315                 return;
316
317         while (*cp) {
318                 tty_insert_flip_char(tty, *cp, 0);
319                 cp++;
320         }
321         con_schedule_flip(tty);
322 }
```

Funciones

- **applkey** : Escribe el código de la tecla pulsada en el buffer.

```
324 static void applkey(struct vc_data *vc, int key, char mode)
325 {
326         static char buf[] = { 0x1b, 'O', 0x00, 0x00 };
327
328         buf[1] = (mode ? 'O' : '[');
329         buf[2] = key;
330         puts_queue(vc, buf);
331 }
```

Otras funciones

- **fn_caps_toggle**: intercambia el estado del LED Bloq. Mayús
- **fn_caps_on**: Activa el LED de Bloq. Mayús
- **fn_enter**: Introduce en el buffer el carácter retorno de carro y también puede añadir nueva linea.
- **fn_num/fn_bare_num**: Cambia el LED de Bloq.Num.
- **handle_diacr**: Para combinar teclas especiales como ‘¨’ o ‘´’ con una tecla ordinaria para obtener caracteres especiales.

Otras funciones

- **fn_boot_it**: Se usa para enviar a consola la combinación: Ctrl+Alt+Supr.

```
575  static void fn_boot_it(struct vc_data *vc)
576  {
577          ctrl_alt_del();
578  }

900  void ctrl_alt_del(void)
901  {
902          static DECLARE_WORK(cad_work, deferred_cad);
903          //Si la variable C_A_D esta activada se reinicia
904          if (C_A_D)
905                  schedule_work(&cad_work);
906          else            //si no se envia la señal SIGINT al proceso activo.
907                  kill_cad_pid(SIGINT, 1);
908  }
```

Otras funciones

- **fn_compose**: Se utiliza para detectar combinaciones de teclas que producirán caracteres especiales, como la cedilla (ç,Ç).
- 580 static void fn_compose(struct vc_data *vc)
- 581 {
- 582 dead_key_next = 1;
- 583 }

Otras funciones

- **k_unicode**: Inserta teclas muertas en el buffer.

```
639 static void k_unicode(struct vc_data *vc, unsigned int value, char up_flag)
640 {
641         if (up_flag)
642                 return;
643
644         if (diacr)
645                 value = handle_diacr(vc, value);
646
647         if (dead_key_next) {
648                 dead_key_next = 0;
649                 diacr = value;
650                 return;
651         }
652         if (kbd->kbdmode == VC_UNICODE)
653                 to_utf8(vc, conv_8bit_to_uni(value));
654         else if (value < 0x100)
655                 put_queue(vc, value);
656 }
```

Otras funciones

- **k_dead2** : Combina teclas muertas con la siguiente tecla.

```
675  static void k_dead2(struct vc_data *vc, unsigned char value, char up_flag)
676 {
677         k_deadunicode(vc, value, up_flag);
678 }

663 static void k_deadunicode(struct vc_data *vc, unsigned int value, char up_flag)
664 {
665         if (up_flag)
666                 return;
667         diacr = (diacr ? handle_diacr(vc, value) : value);
668 }
```

Otras funciones

- **k_fn**: Permite usar las teclas de función.

```
697 static void k_fn(struct vc_data *vc, unsigned char value, char up_flag)
698 {
699         unsigned v;
700
701         if (up_flag)
702                 return;
703         v = value;
704         if (v < ARRAY_SIZE(func_table)) {
705                 if (func_table[value])
706                         puts_queue(vc, func_table[value]);
707         } else
708                 printk(KERN_ERR "k_fn called with value=%d\n", value);
709 }
```

Otras funciones

- **k_cur**: Con esta función se puede usar los cursores.

```
711 static void k_cur(struct vc_data *vc, unsigned char value, char up_flag)
712 {
713         static const char cur_chars[] = "BDCA";
714
715         if (up_flag)
716                 return;
717         applkey(vc, cur_chars[value], vc_kbd_mode(kbd, VC_CKMODE));
718 }
```

Otras funciones

- **k_pad**: Si Bloq. Núm. esta activado insertara en el buffer los números.
- **k_shift**: Inserta la combinación Shift + Tecla.
- **k_meta**: Inserta la combinación Alt + Tecla.
- **k_ascii**: Combina la tecla Alt+ código numérico para producir el carácter correspondiente.

Otras funciones

- **getledstate**: Devuelve el estado de los LED del teclado.
- **setledstate**: Se especifica el estado de los LED del teclado.
- **getleds**: Devuelve los LED presentes en el teclado.